MX-101 オペレーションマニュアル ■取扱説明書 ■BASIC入門 ■リファレンス CASIO

# MX-101
# OPERATION MANUAL

■取扱説明書 ■BASIC入門 ■リファレンス

CASIO®

MSX

# MX-101
# OPERATION MANUAL

■取扱説明書■BASIC入門■リファレンス

CASIO®

## はじめに

このたびは、CASIO MX-101をお買い上げいただきまして、誠にありがとうございます。お求めのMX-101を正しくご使用いただくため、この「OPERATION MANUAL」をよくお読みください。また、一度お読みになったあとも、いざという時にお役に立つよう必ず保管しておいてください。

本書は、大きくわけて次の3部構成になっています。

**第1部　取扱説明書**

本機を正しく使うための基本的なことが書いてあります。必ずお読みください。

**第2部　BASIC入門**

パーソナルコンピュータは、他の家庭用電気製品とちがって、プログラムによって動きます。プログラムは市販もされていますが、ここではBASICが初めての人、勉強したい人のために、わかりやすくMSX BASICの解説をしています。

**第3部　リファレンス**

MSX BASICの機能を詳細に解説してあります。

ここに書かれていることを全て暗記する必要はありません。必要に応じて、必要なページを辞書を引くときのような気持ちで読んでください。

- 本書の内容に関しては、将来予告なしに変更することがあります。
- 本書の内容については万全を期して作成いたしましたが、万一不審な点や誤りなど、お気付きのことがありましたらご連絡ください。
- 本書使用による損害および逸失利益等につきましては、当社では一切その責任を負えませんので、あらかじめご了承ください。
- 
- 万一、本機使用により生じた損害、逸失利益または第三者からのいかなる請求についても、当社では一切その責任を負えませんので、あらかじめご了承ください。
- 故障、修理、電池交換等に起因するデータ・プログラムの消失による、損害および逸失利益等につきましては、当社では一切その責任を負えませんので、あらかじめご了承ください。

  なお、大切なデータおよびプログラムはカセットテープやディスクシートに記録しておいたり、また控えのテープやディスクシートを作っておくことをおすすめします。

目　次

# 第1部　取扱説明書

目　次

# 第2部 BASIC入門

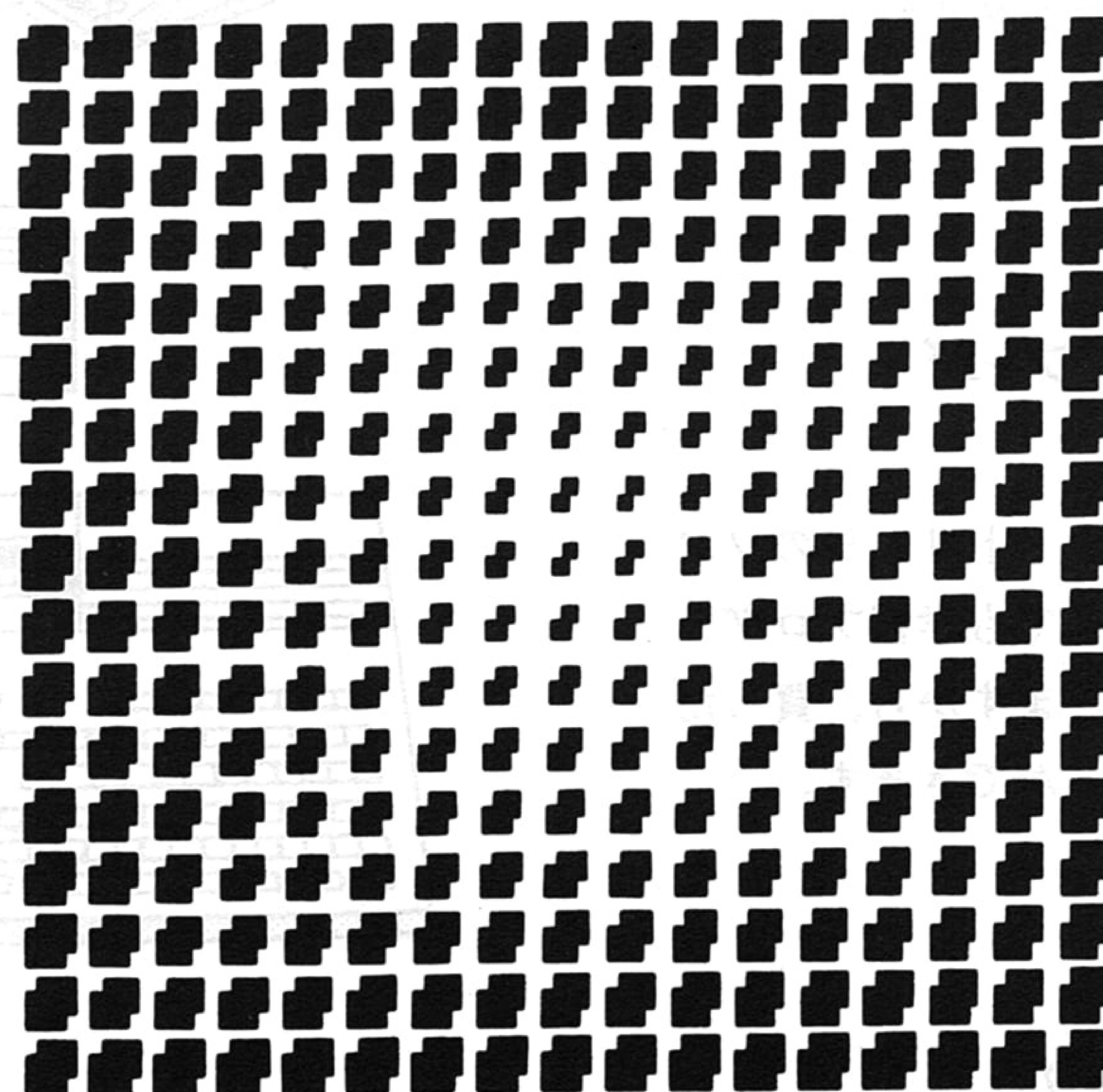
## 第7章　プログラムを楽しもう……104

目 次

# 第3部 リファレンス

# 第1部

# 取扱説明書



# 第1章　MX-101の紹介

CASIO MX-101は、MSX規格のホームコンピュータです。このMSX規格は、メーカーや機種を問わないパーソナルコンピュータの統一規格ですから、お友だちにMSXを使っている人がいたら、その人の使っているゲームやプログラムは、このMX-101にも使えるので、大変便利です。では、順を追ってMX-101の特長を説明しましょう。

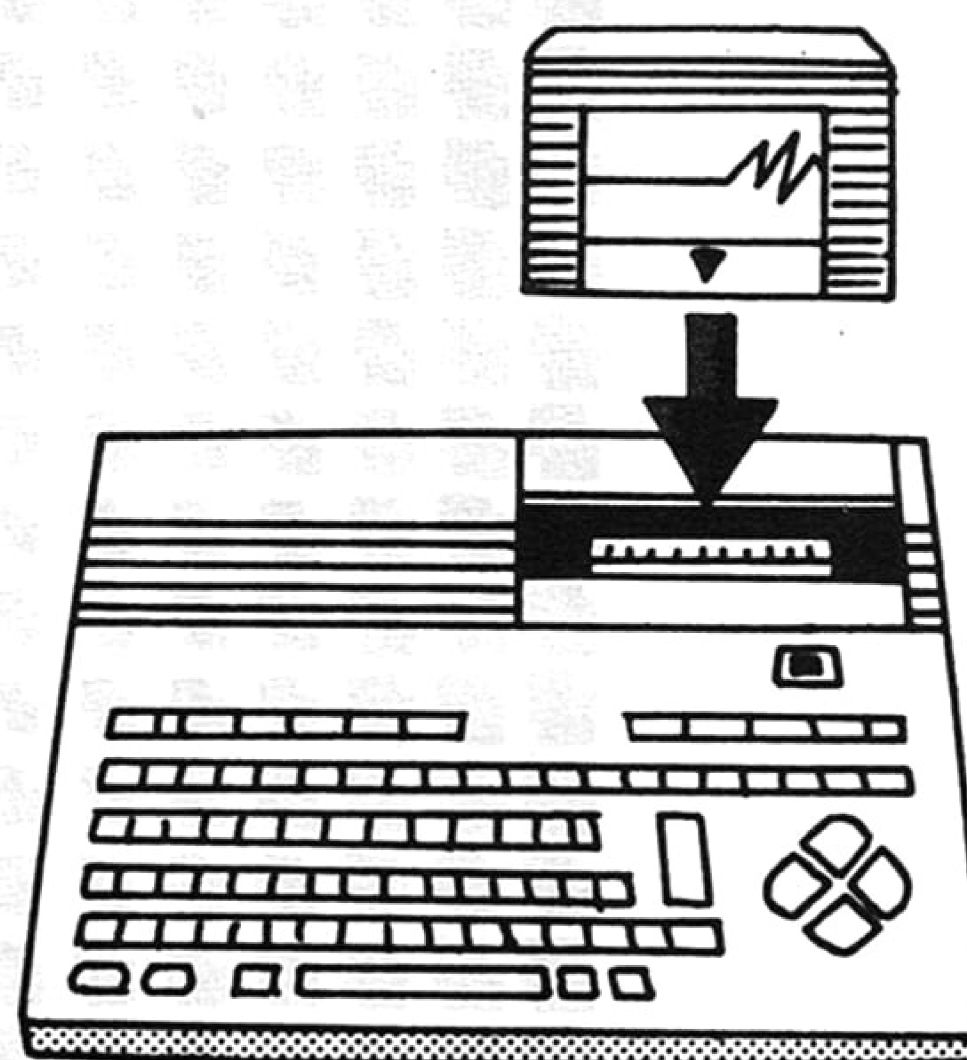
## ■MSX　BASICを内蔵

本機は、数多くの命令を持ったＭＳＸ BASICを内蔵していますから、高度な計算や、音楽、グラフィックなどもでき、パソコンとして初心者の人からベテランの人まで幅広く使えます。
もちろん、他のＭＳＸ機種で作られたプログラムも本機で活用することができます。



## ■MSXのカートリッジ・ソフトが使えます。

ＭＳＸ用に発売されているカートリッジのソフトならば、ゲームプログラムでも学習プログラムでも自由に使うことができますから、買ったその日からすぐに楽しむことができます。





## ■3重和音の出せる音楽演奏機能

楽しく音楽を演奏するため、８オクターブ３重和音で、本格的な音楽演奏を行なうことができます。
もちろんテレビゲームなどのような効果音も自由に出すことができます。

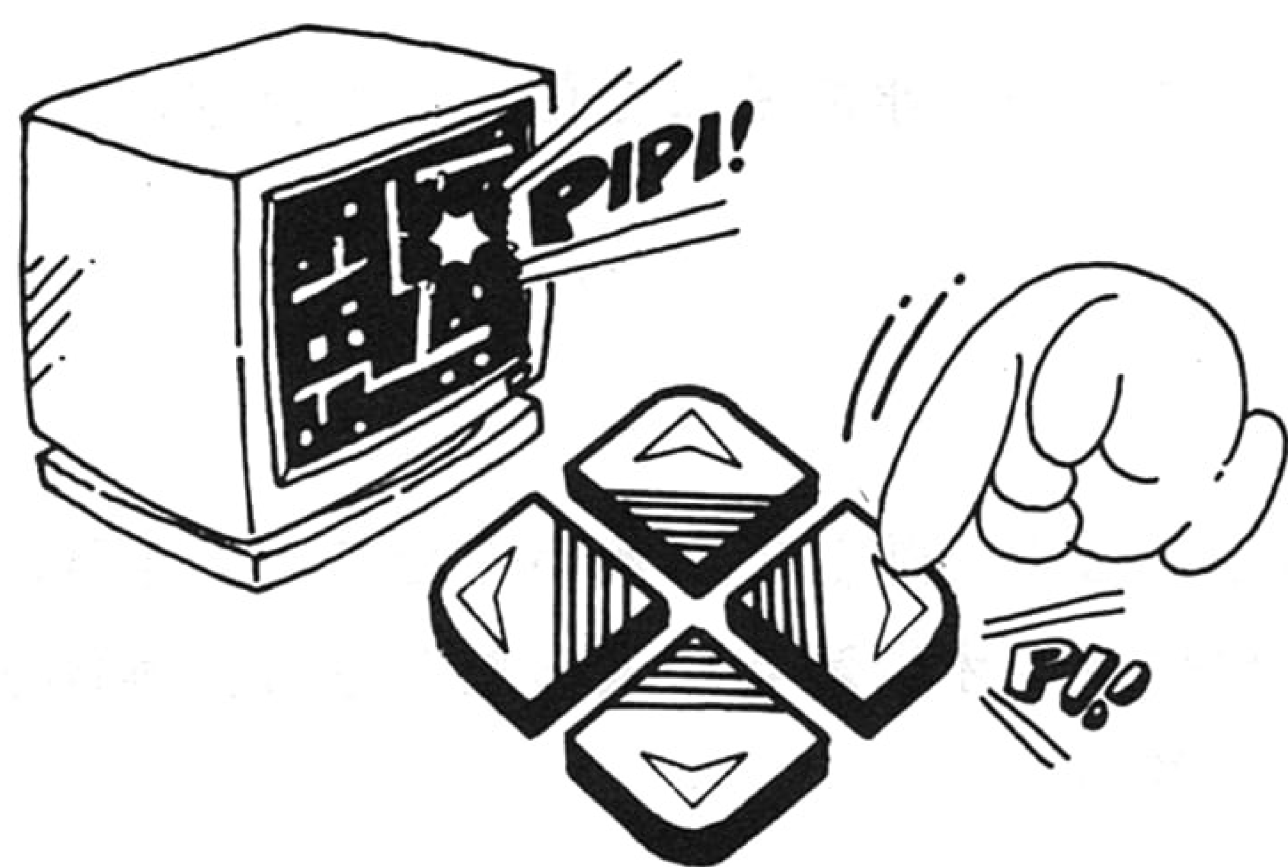
## ■多彩なカラー表示

本機は16色のカラー表示ができます。また、スプライト機能といって、キャラクター（人物やミサイル等）を画面上で自由に動かすことができます。



## ■はじめからついているジョイパッド・トリガーキー

ゲームをするときに使うジョイスティックの代りに、本機ではキーボード上にジョイパッドと、トリガーキーというキーがついています。ですから、ジョイスティックがなくてもすぐにゲームを楽しむことができます。
(ジョイスティック１のかわり)




## ■コードレスで家庭のテレビが使えます。

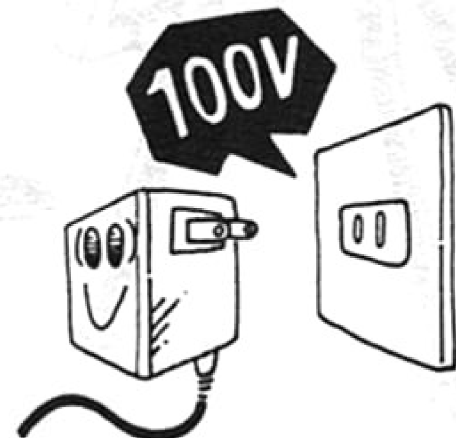
UHFチャンネルのついている家庭用テレビなら、MX-101から送信するUHF電波を、専用の受信アンテナを接続して、コードレスで使えます。また、音声・映像入力端子がついているAVテレビを使うことができます。

**（注意）**
本機のRAM容量は、16Kバイトです。市販のソフトウェア（ROMカートリッジ、カセットテープなど）を購入するときは、必ずそのソフトウェアを使用するのに必要なメモリーサイズを確認してください。
なお、MX-101のメモリー増設については258Pを参照ください。

# 第2章　取扱い上の注意

本機の使用にあたっては、以下のことに注意してください。

## ●電源について

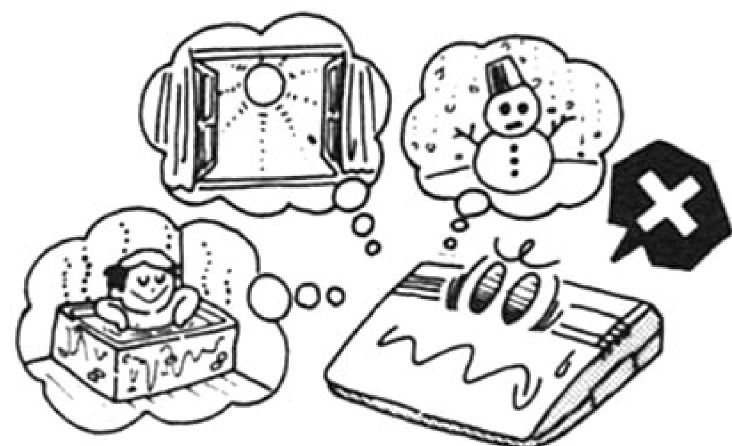



①ACアダプターは家庭用電源コンセント（AC100V）につなぎます。　(外国では使用できません)




②ACアダプターのコードは無理に曲げたり、コードの上に重い物を乗せたりしないでください。

③傷がついたコードは絶対に使わないでください。



④温度の高いところや、水のそばで使うことは避けてください。



⑤テレビと本体とは少し離してお使いください。本体の近くにあるテレビやラジオに雑音が入ったときは、本体をもっと離してください。



⑥本体が汚れた場合は、シンナーやベンジンなどを使わずに、「よくかわいたやわらかい布」か「中性洗剤に浸し、よく絞った布」でふいてください。



⑦電源を抜くときにはコードをひっぱらず、プラグをしっかり持って抜いてください。

⑧もし、異常がおきたら電源を切って、ACアダプターをコンセントからはずしてください。



⑨長い間使わない場合は、ACアダプターをコンセントから抜いてください。

⑩送信用アンテナ、受信用アンテナは、MX-101を使う目的以外に使用しないでください。また、アンテナで目や手など突かないよう注意してください。




⑪本体の左上にある通風孔はふさがないでください。

⑫乱暴に扱ったり、本体の上に物を乗せたりしないでください。



⑬絶対分解しないでください。

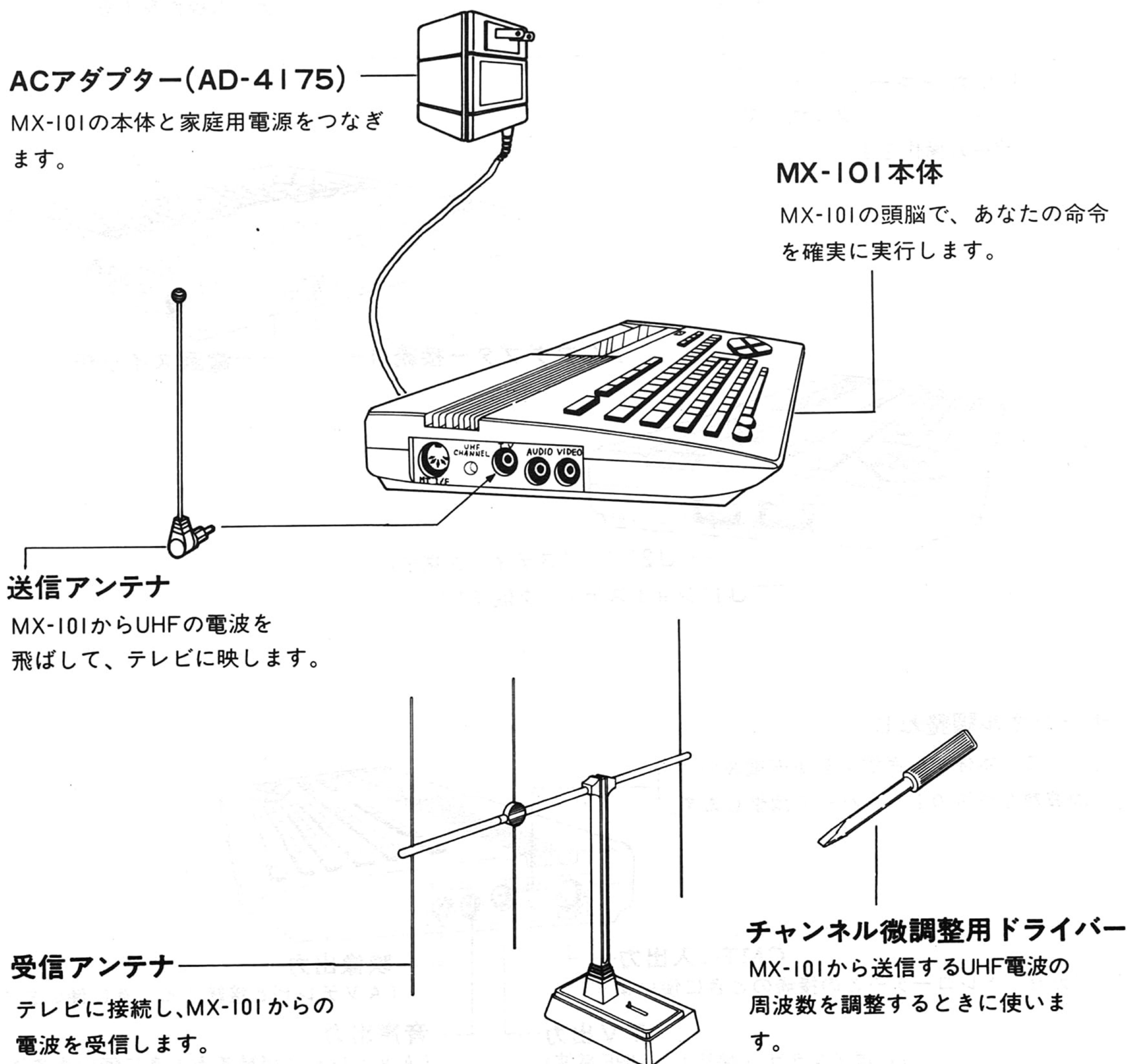
| 使用中おかしなことがあったら、第７章のトラブルシューティングのところを見てください。 |
|---|

# 第3章 MX-101の各部の名称と接続方法

## 3-1 MX-101のセット内容

MX-101には、次のものが入っています。内容を確かめながら、それぞれの名称を覚えてください。

■MX-101の本体、ACアダプター、送信アンテナ、受信アンテナ、 チャンネル微調整用ドライバー、オペレーションマニュアル、保証書



# 3-2 各部の名称

**スロット**(カートリッジをさし込むところ)

※本体のスロットには、±12Vを配線していません。このため、±12Vを使用するインターフェイスカートッジ(市販のRS-232C等)は、拡張BOX(KB-10)のスロットに挿入してください。

**電源ランプ**

**キーボード**

**ジョイパッド**
(ジョイスティックに代ってゲーム操作をする)

**トリガーキー**
(ジョイスティックに代ってゲーム操作をする)

**ACアダプター接続口**

**電源スイッチ**

**J2**(ジョイスティック端子2)

**J1**(ジョイスティック端子1)

**チャンネル調整ねじ**
(ここで、本体から送信するUHF電波の周波数を付属のドライバーで調整します。)



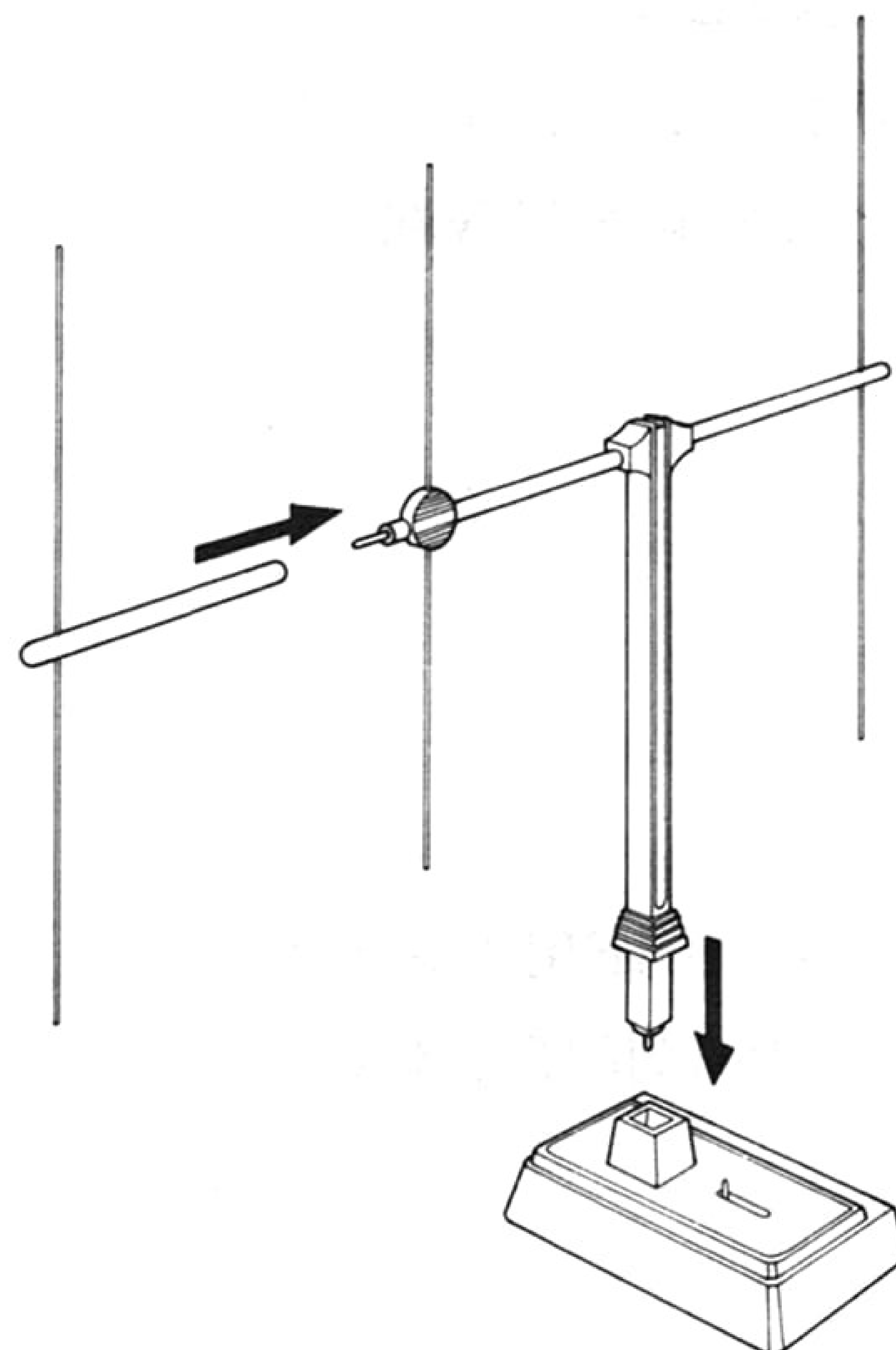
**CMT I/F 入出力**
(カセットレコーダーとの接続のときに使います)

**映像出力**
(AVテレビと接続するときに使います)

**TV出力**
(送信アンテナを接続し、UHF電波をテレビに送信します。)

**音声出力**
(AVテレビと接続するときに使います)

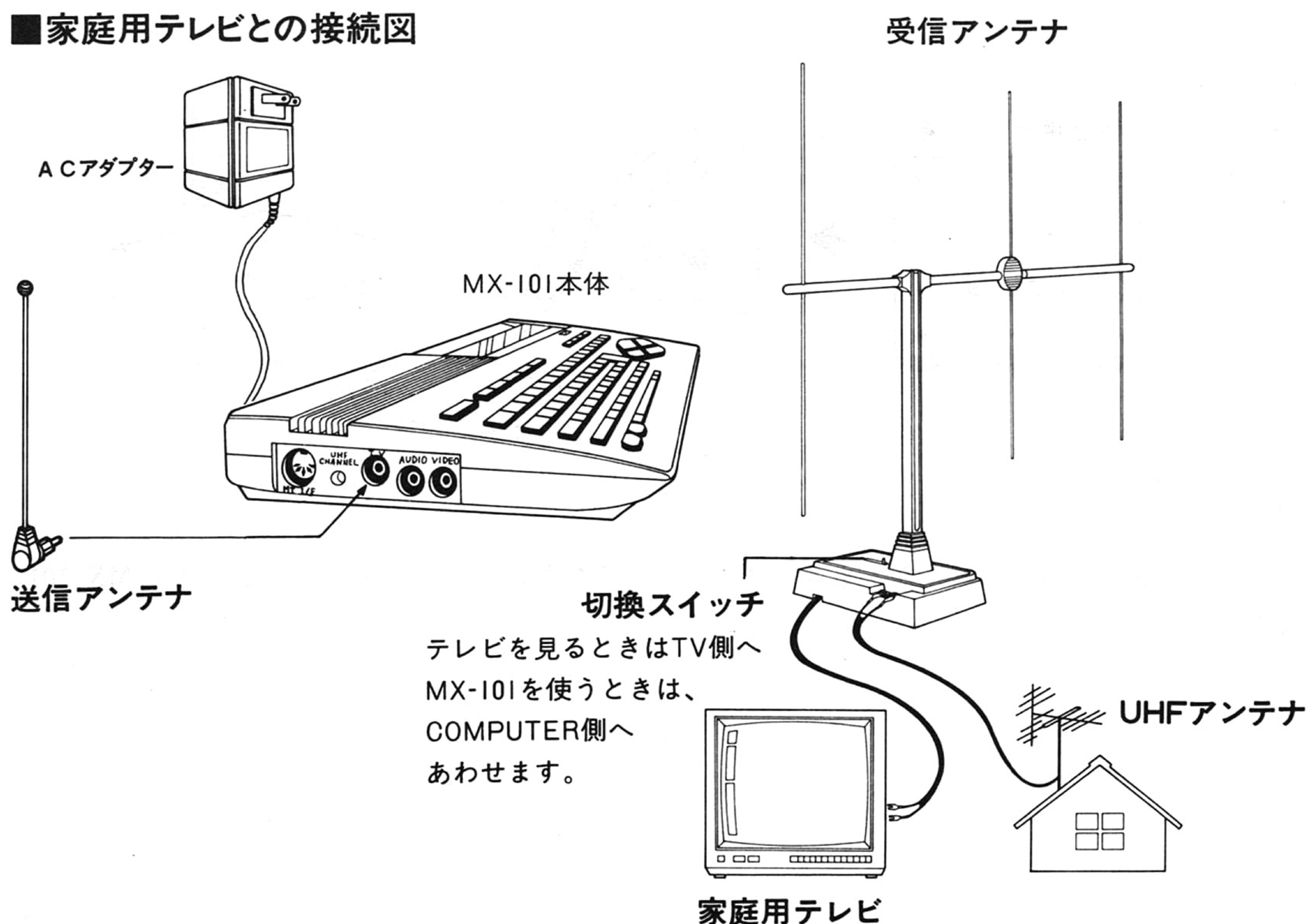
## ■受信アンテナの組立方



# 3-3 本体とテレビの接続

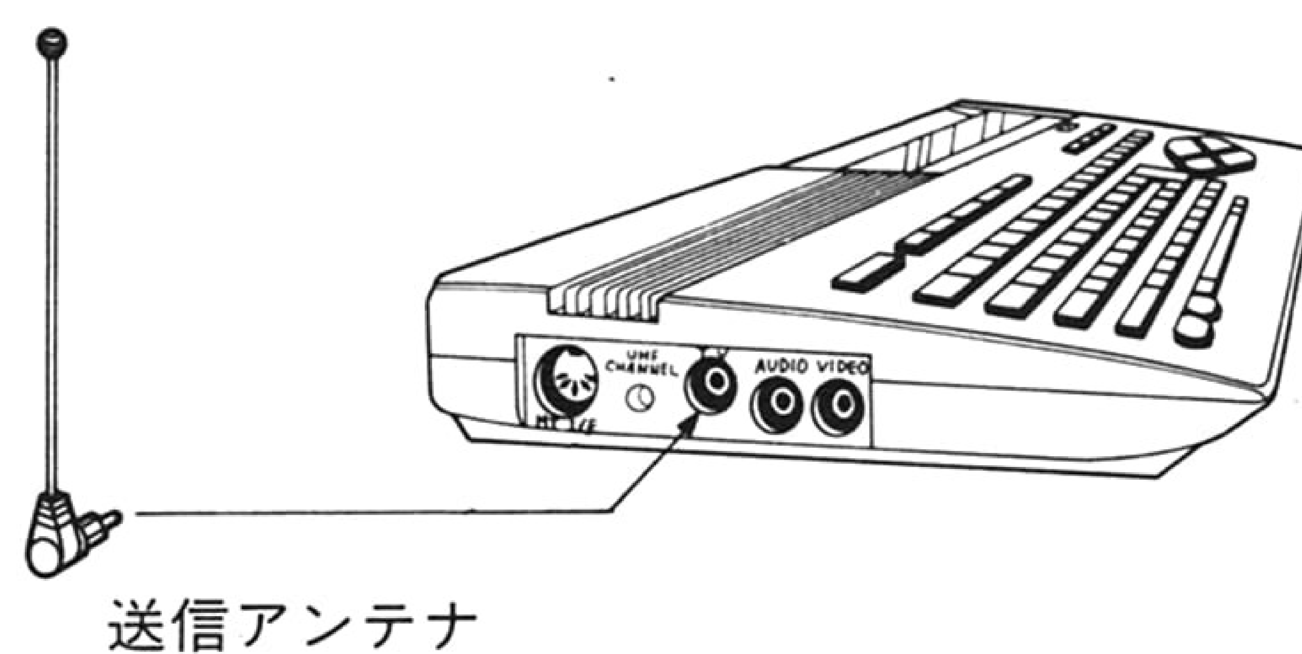
さあ、これからMX-101と本体を接続します。
MX-101をスムーズに動かすために、説明にしたがって正しく接続してください。テレビには、UHFチャンネルのついた家庭用テレビと、映像音声入力端子のついたAVテレビがあり、どちらも使えますが、接続のしかたがそれぞれが違うので注意してください。(UHFチャンネルのついていない家庭用テレビでは使用できません)

## ■家庭用テレビとの接続図

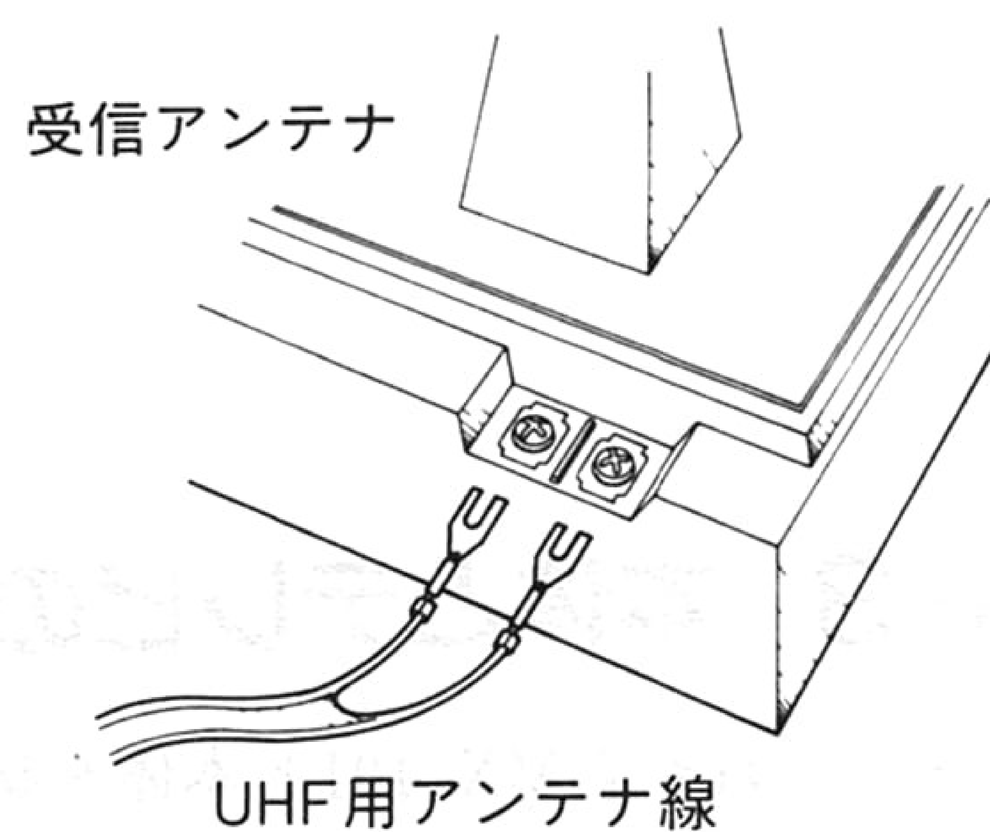


## ■家庭用テレビでの接続のしかた

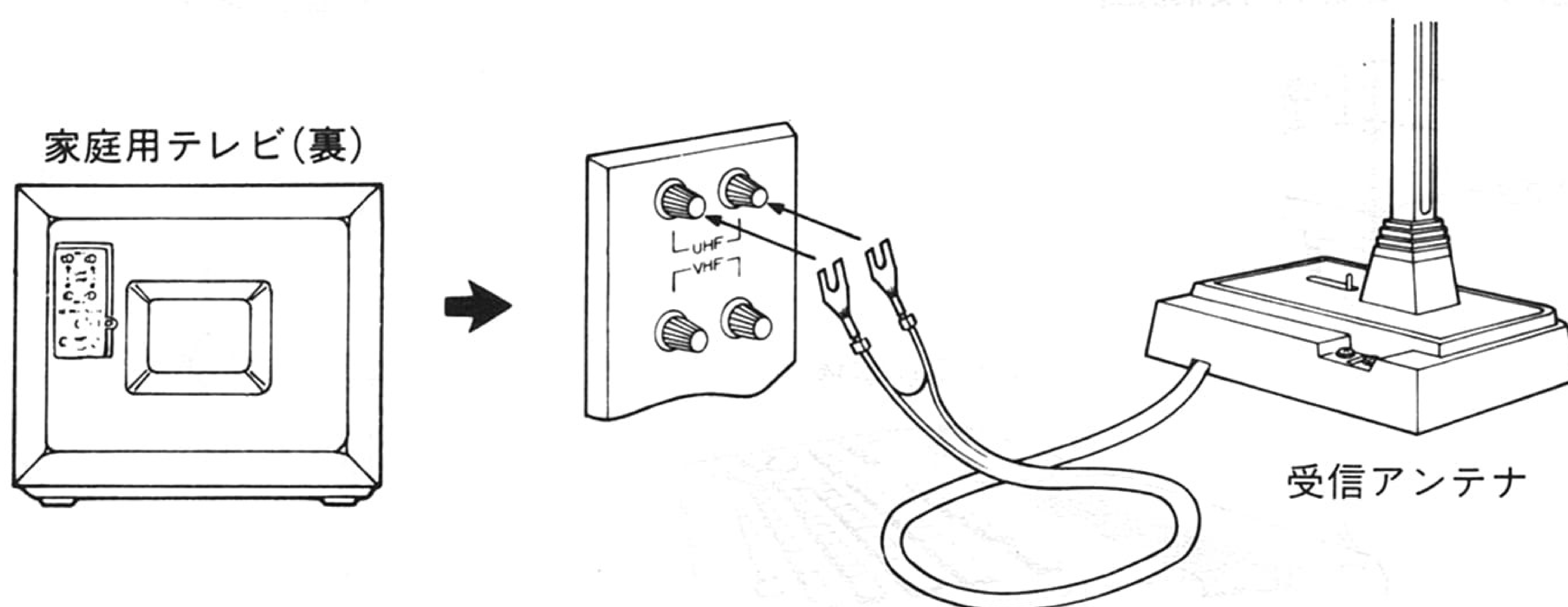
①MX-101本体の電源がOFFになっていることを確認し、「送信アンテナ」を本体のTV端子に接続します。




②いままでTVに接続していたUHF用アンテナ線をはずし、「受信アンテナ」に接続します。



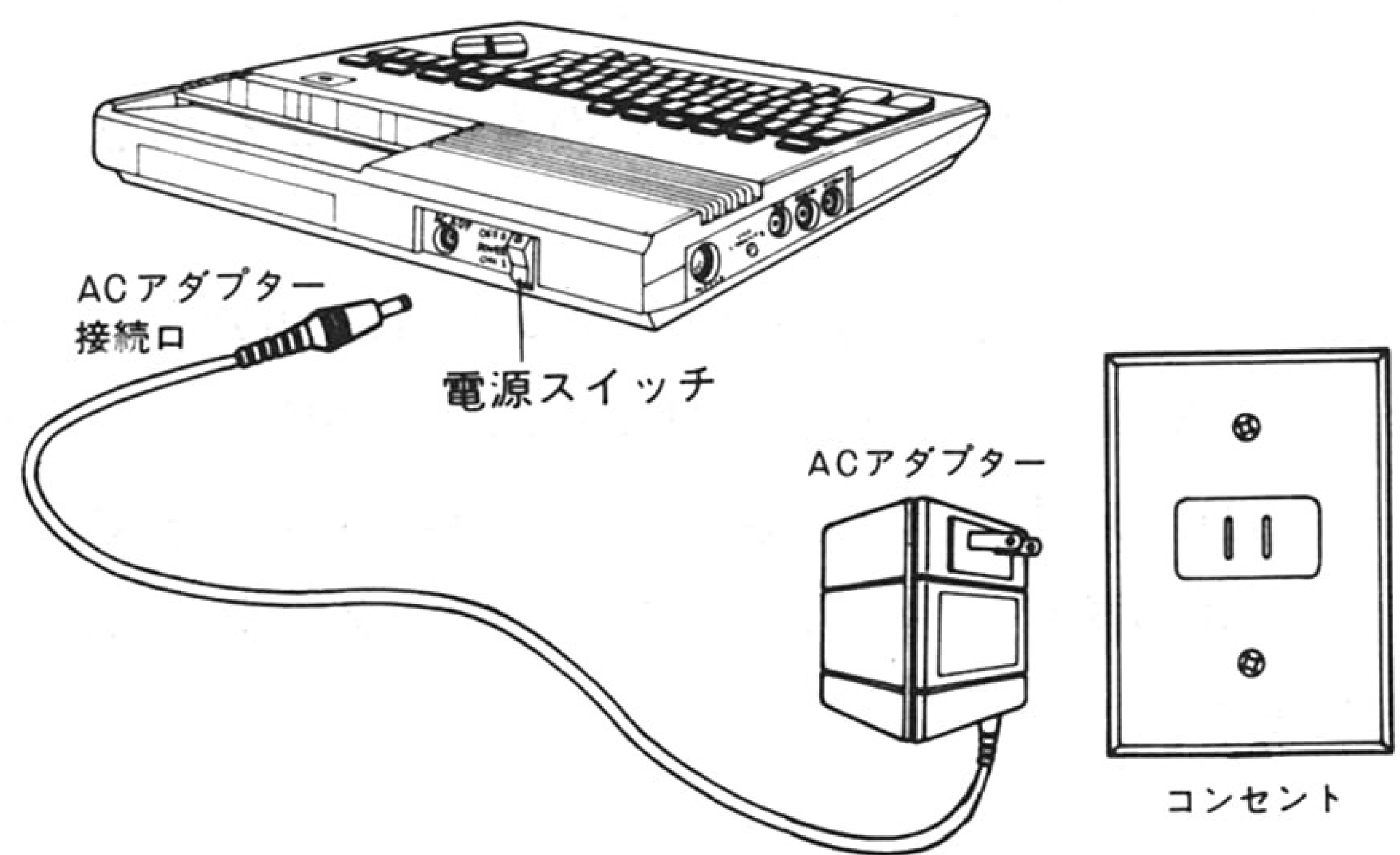

③「受信アンテナ」から出ているアンテナ線を、TVのUHF用アンテナ端子に接続します。




上記の接続が正しく行なわれたか、もう一度確認してください。正しく接続されていたら、MX-101の画像を受信しましょう。

## ■受信のしかた

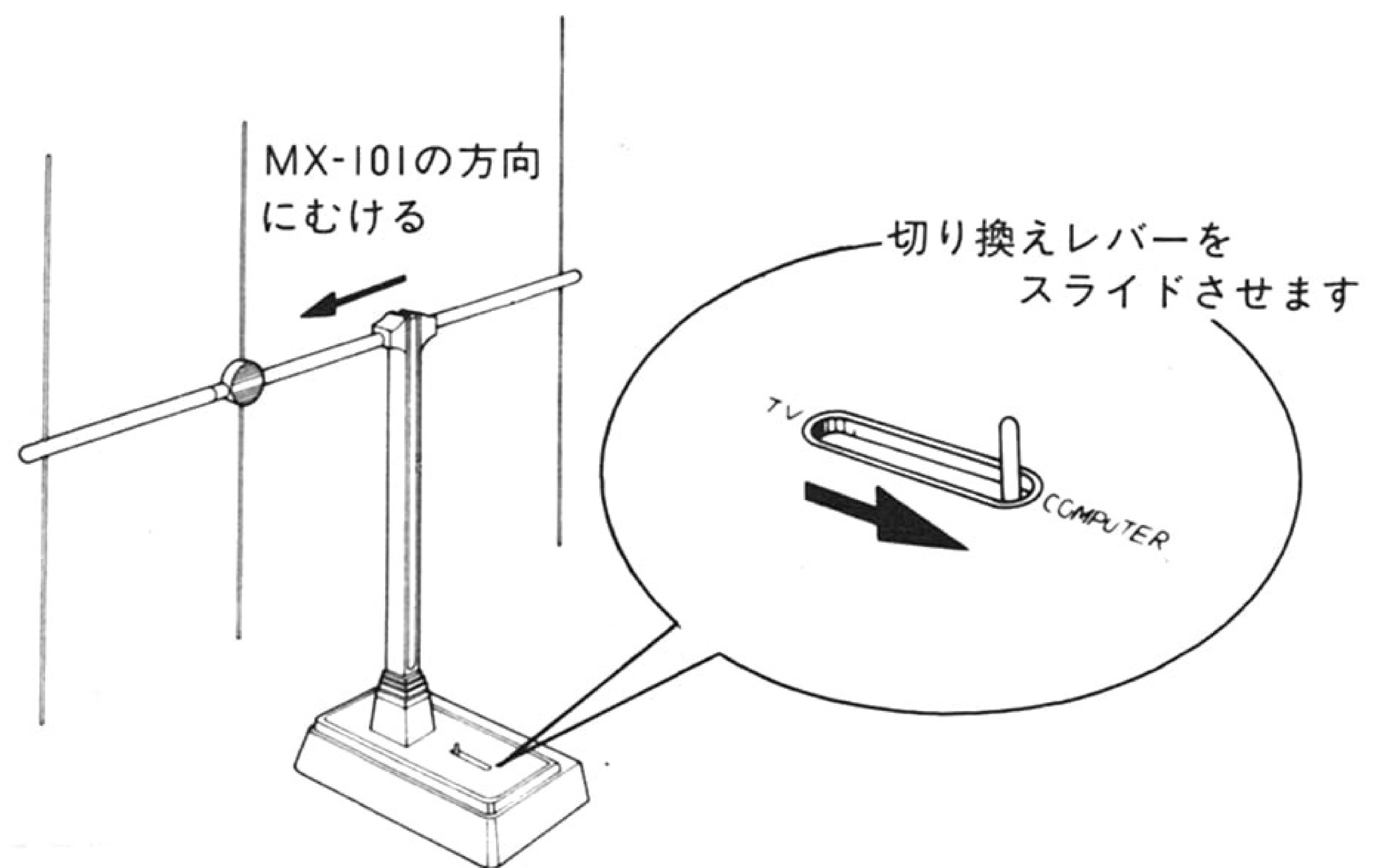
①本体の電源スイッチがOFFの状態で、ACアダプターと本体をつなぎます。次にACアダプターをコンセントに差し込みます。そして、最後に本体の電源をONにします。




②テレビの電源をONにします。チャンネルをUHFの13チャンネルまたは14チャンネルのあきチャンネル(テレビ放送のないチャンネル)にあわせます。

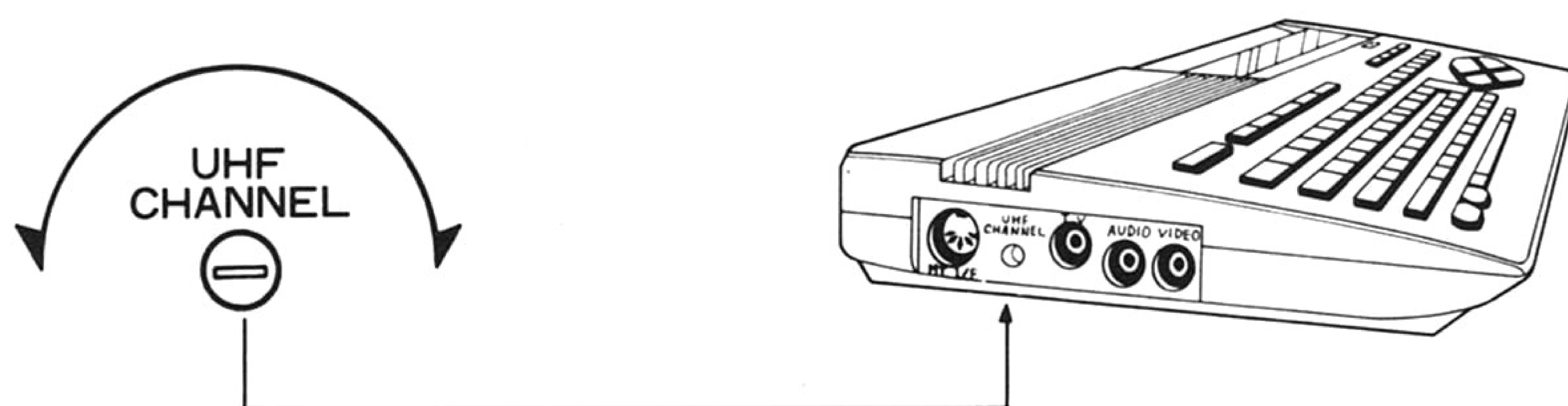
③「受信アンテナ」のCOMPUTER↔TVの切り換えレバーをCOMPUTER側にスライドさせます。また、受信アンテナをMX-101の方向に向けます。

※受信用アンテナとMX-101のあいだに障害物を置かないようにしてください。




④付属のドライバーを、本体の「UHF CHANNEL」つまみに差し込み、左右にまわして、テレビの画像が一番きれいに映るように調整します。

※送信・受信アンテナに手で触れると、画像が乱れる場合があります。




**ご注意**

MX-101の送信するUHF電波の周波数は、470.25MHz～478.25MHzです。テレビのチャンネルを13chまたは14chにあわせて、本体のUHF CHANNEL」つまみを調整をしても本体の画像が映らないときは、テレビ側で周波数を調整し、MX-101の送信している周波数と同調させてください。

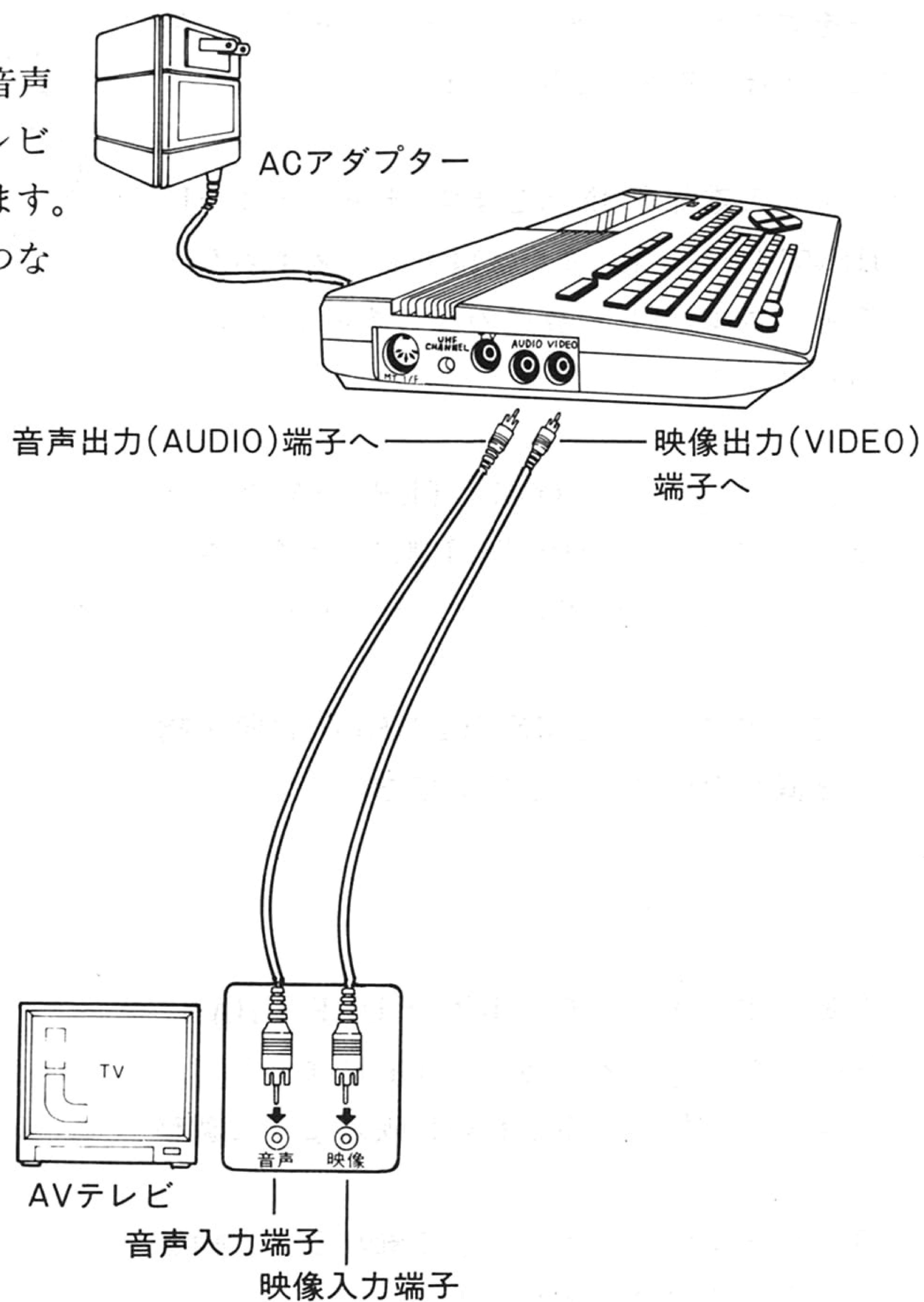
※テレビの周波数の調整のしかたについては、お手持ちのテレビの取扱説明書をお読みください。

## ■AV（音声・映像入力端子付）テレビに接続するときは

AVテレビに接続するときは、市販のピンコード（コードの両端がピンプラグのもの）を2本用意してください。

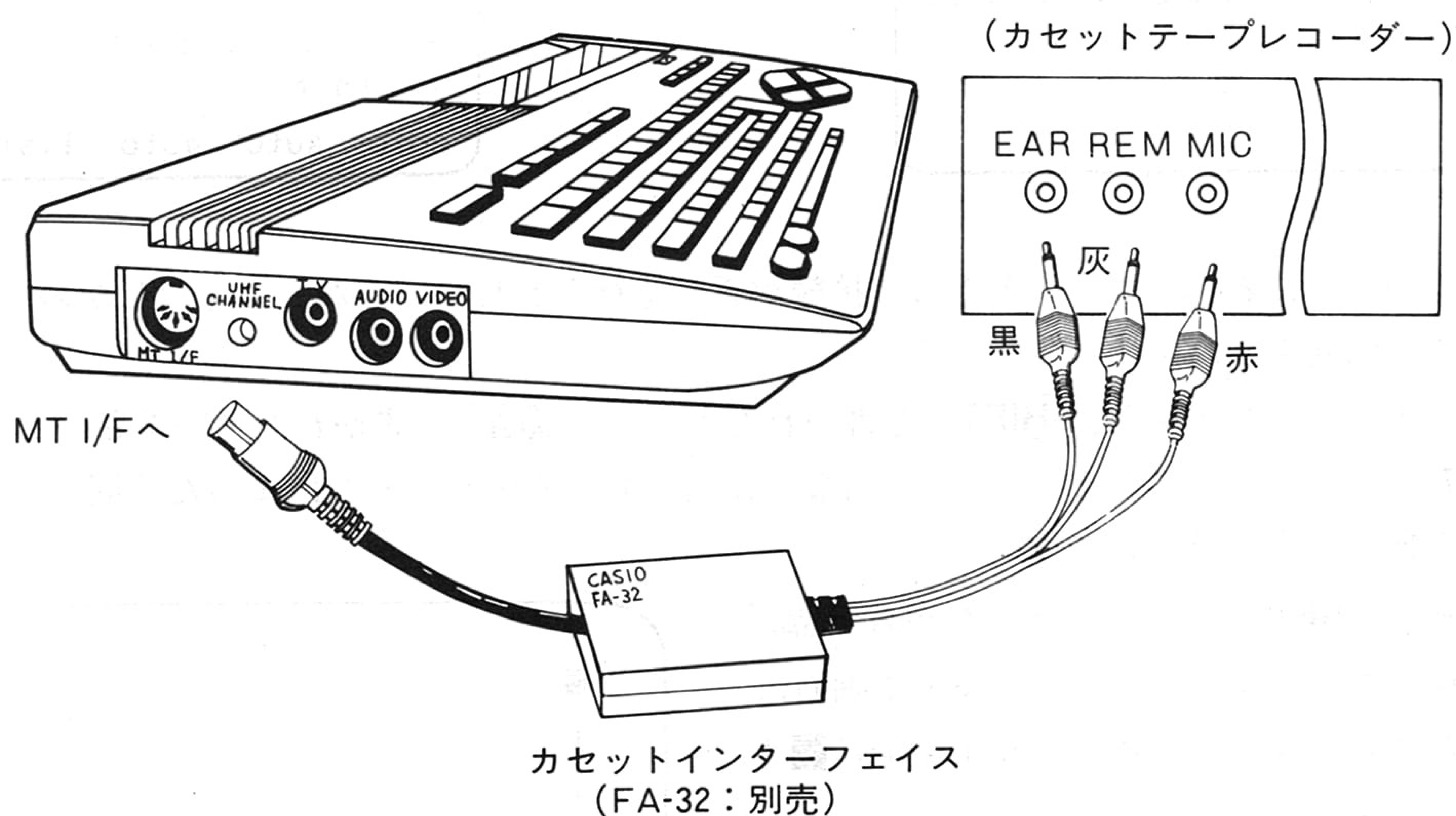
MX-101のAUDIO端子とAVテレビAUDIO（音声入力）端子、MX-101のVIDEO端子とAVテレビのVIDEO（映像入力）端子をそれぞれ接続します。

コードの接続が終ったら、ACアダプターをつなぎます。



# 3-4 カセットテープレコーダーとの接続の仕方

〝カセットテープレコーダーは、プログラムやデータの保存やテープのゲームを行なう時に使います。それ以外は必要ありません。〟
カセットテープレコーダーをMX-101本体に接続するときは、MX-101本体の電源は、、必ずOFFにして行なってください。カセットテープレコーダーは、みなさんが現在お使いになっている普通のものと、パソコン専用のテープレコーダー(例：カシオデータレコーダーKR-10)があります。
※ほとんどのカセットテープレコーダーが使えますが、一部使えないものもあります。




EAR———出力(イヤホン)… 黒色プラグ
REM———リモートコントロール…灰色プラグ
MIC———入力(マイク)… 赤色プラグ

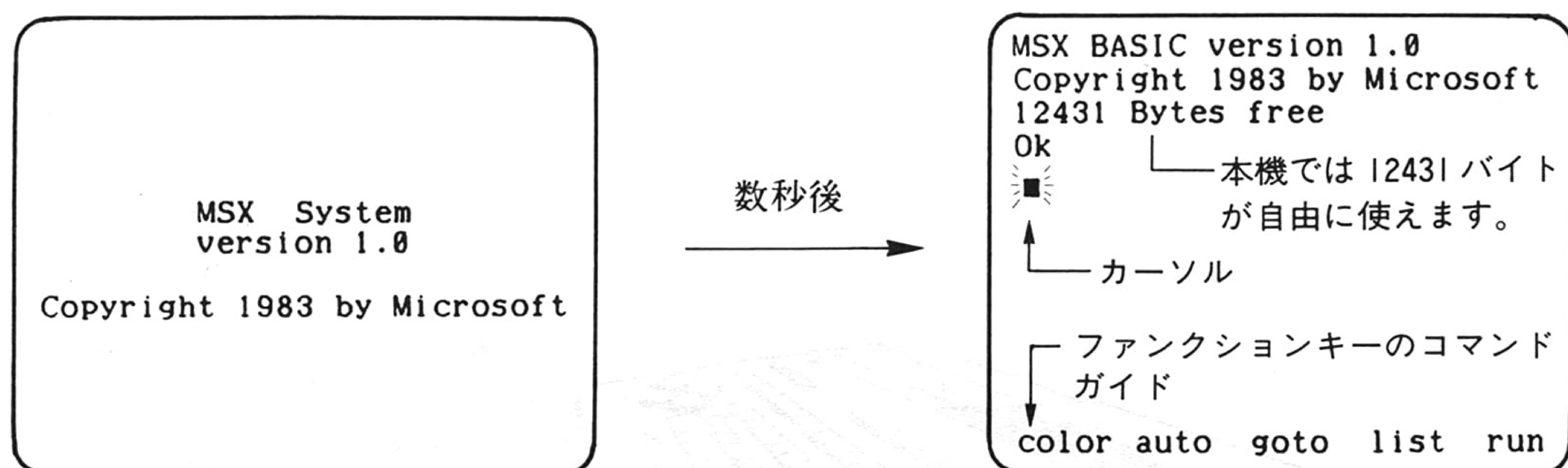
注) リモート端子のないカセットテープレコーダーの場合は、リモートプラグはどこにもつなぎません。

# 3-5 作動開始

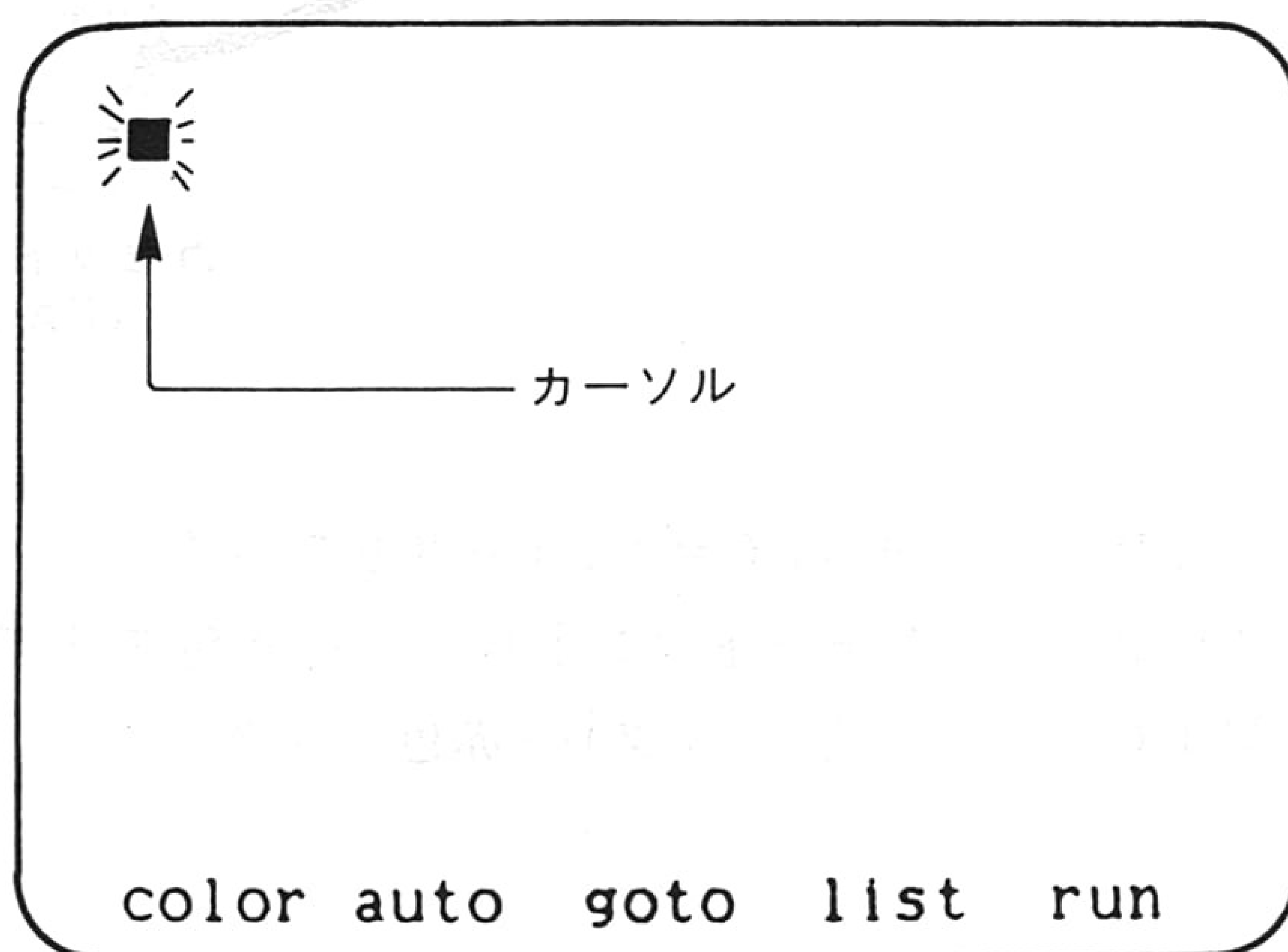
パソコンとテレビとの接続が終ったら、次の順序でスイッチを入れてください。

①テレビ、パソコンの電源プラグをコンセントに差し込みます。
②テレビのスイッチを入れます。
③パソコンのスイッチを入れます。
※消す場合も、テレビのスイッチを切ってから、パソコンのスイッチを消してください。

間もなく、テレビ画面が左下のようになり、数秒たつと右下のようになります。




画面が上のように表示されないときは、接続が正しく行なわれているか、よくたしかめてください。
次に、この画面をきれいにしましょう。
まずキーボードのなかから SHIFT と書かれたキーと、CLS HOME と書かれたキーをさがしてください。
SHIFT キーは、キーボードの左端と右端のふたつありますが、働きはまったく同じですから、好きな方を使ってください。
では、その SHIFT キーを押しながら CLS HOME キーを押してください。すると、今まで画面に表示されていた文字が消えて、左上の隅に■が表示されます。これをカーソルといいます。




※以後 SHIFT キーを押しながら CLS HOME キーを押すことを、SHIFT → CLS HOME と書きますので、よく覚えておいてください。

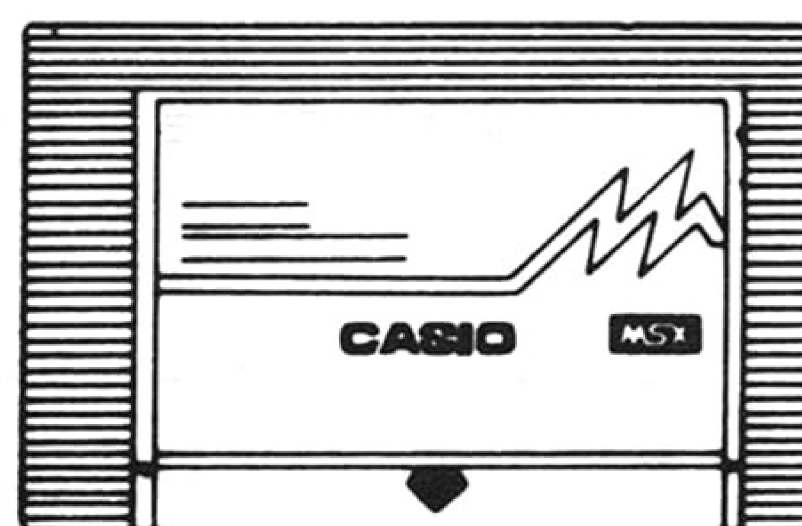
これで準備完了です……!

# 第4章 ROMカートリッジを使ってみよう

## 4-1 ROMカートリッジとは

市販されているMSX規格のゲームソフトには、テープソフトとカートリッジソフトがありますが、このうちのカートリッジソフトが「ROMカートリッジ」です。テープソフトについてはあとで説明するとして、まず、手軽なROMカートリッジについて説明します。

ROMカートリッジは、四角い箱になっていて、下に細長い溝（スリット）があります。
この中には、パソコンゲームを行うための情報がぎっしりつまっています。あとは簡単な操作でゲームが始められます。
楽しいゲームがいっぱいのROMカートリッジで大いに楽しんでください。




## 4-2 操作は簡単

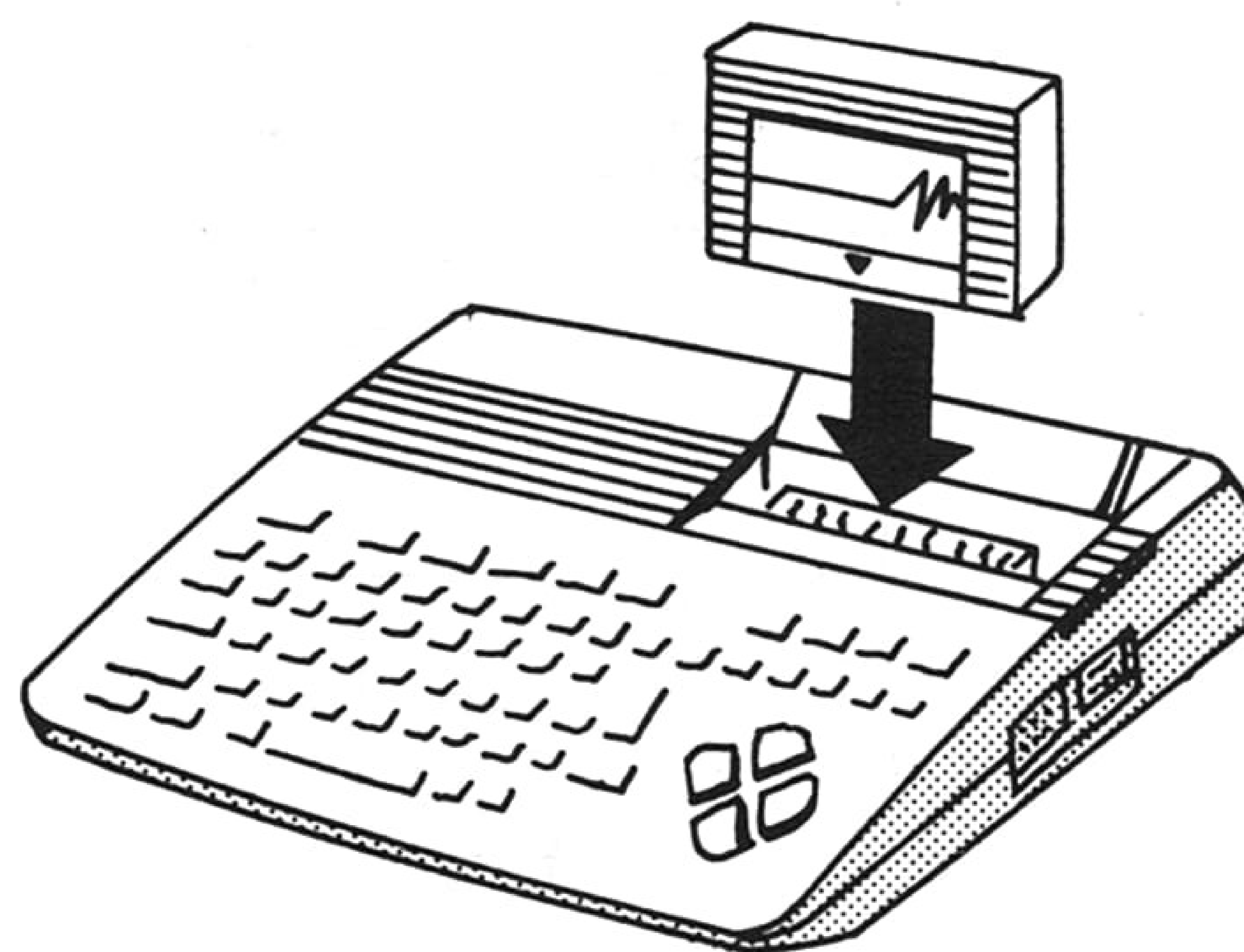
ROMカートリッジが手に入ったら、次のようにしてMX-101に差し込みます。

①パソコンの電源をOFFにします。

②スロットのふたをあけ、ROMカートリッジをスリットのある方を下にして、しっかりと差し込んでください。
このとき、カートリッジの表と裏をよく確かめて、タイトルのラベルが貼ってある方(表)が前にくるように差し込みます。

※なお、カートリッジのスリットにはさわらないでください。

③パソコンの電源スイッチを入れます。
これでOKです。



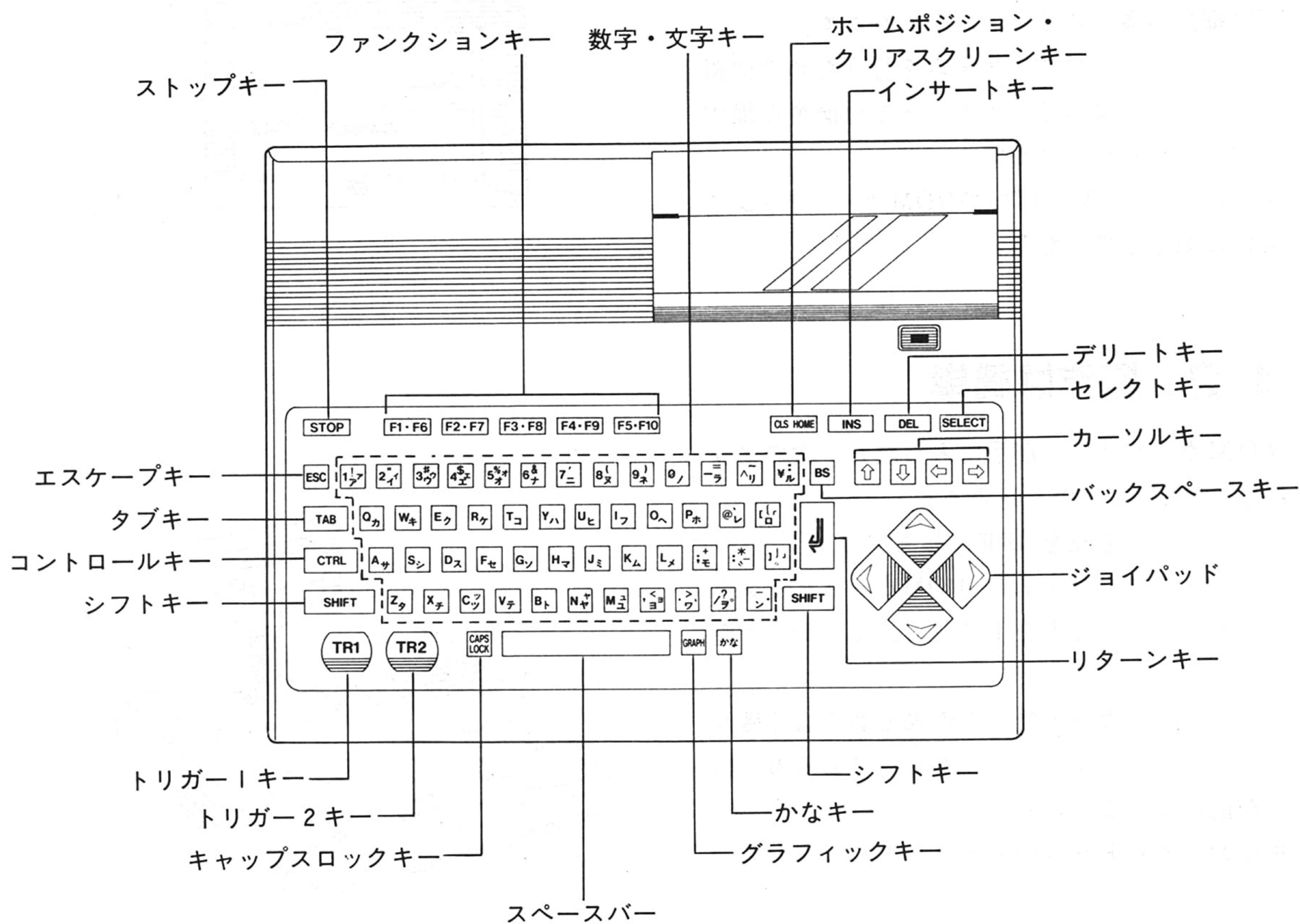
## 4-3 ゲーム開始

カートリッジを差し込んでMX-101のスイッチを入れますと、ゲームのスタート画面が表示されます。
ゲームが始まったら、ジョイパッドやトリガーキーを使って、遊んでください。また、ほとんどの場合、カートリッジの入っていた箱の裏側や中に遊び方の説明がありますので、その説明文もよく読みましょう。
カートリッジをスロットから抜くときは、必ずMX-101の電源スイッチを切ってから行なってください。

# 第5章 キーボードにさわってみよう

## 5-1 キーボードを覚えましょう

キーボードのキーを見てください。ひとつのキーに2つも3つも文字や記号が書かれていますね。MX-101と友だちになるために、キーの名前をしっかり覚えましょう。

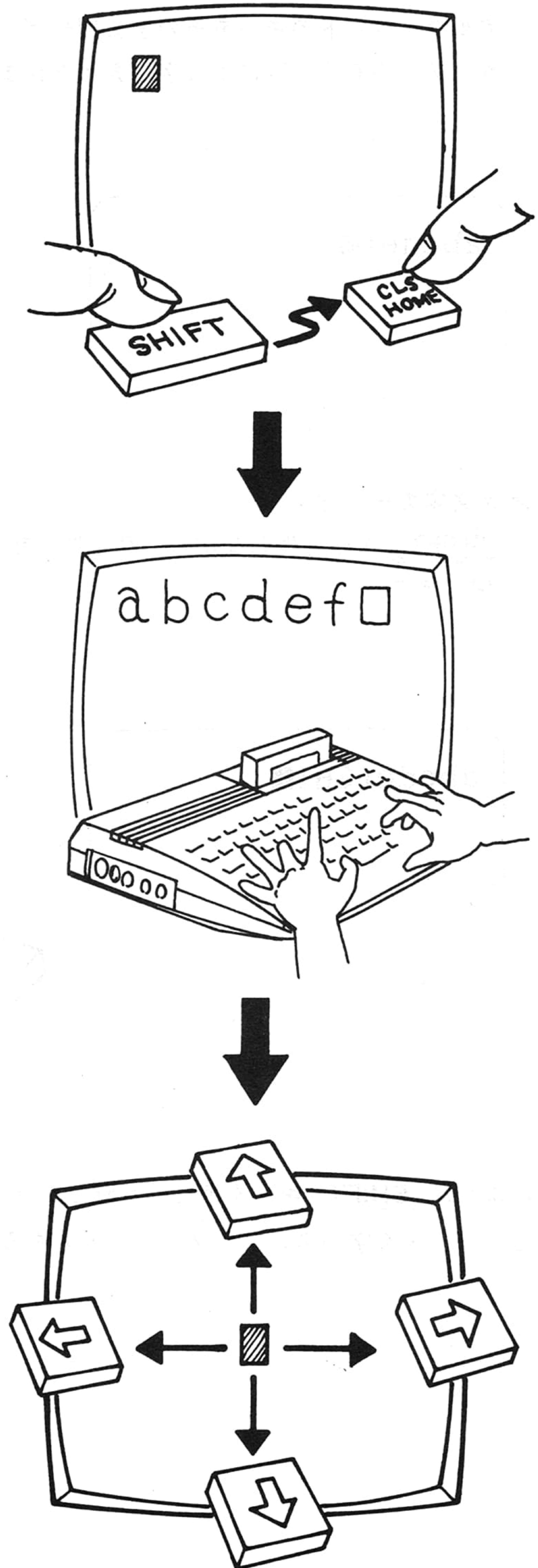


## 5-2 カーソルは自由自在

画面になにか文字が残っていたら消してください。( SHIFT → CLS HOME でしたね)
では、キーを自由に押してください。画面の左上から横書きに、文字や記号があらわれてくるでしょう。ここで気がつくことは、カーソルのあったところに文字や記号が出てくることです。このカーソルを前後左右に動かすのが、キーボードの上部にある4つの矢印、カーソルキーです。ちょっと押してみてください。カーソルが押した矢印の方に動くのが確かめられるはずです。カーソルキーは、一度押すと1文字分、押し続けると何文字分も続けて移動します。
ここで、カーソルを画面のセンターに移動させて文字を出してみてください。

※ジョイパッドはゲームでキャラクターを動かすためのキーで、カーソルの移動には使いません。

※ CLS HOME キー(ホームポジション・クリアスクリーン)を押すとカーソルはホーム位置(左上すみ)に移動します。

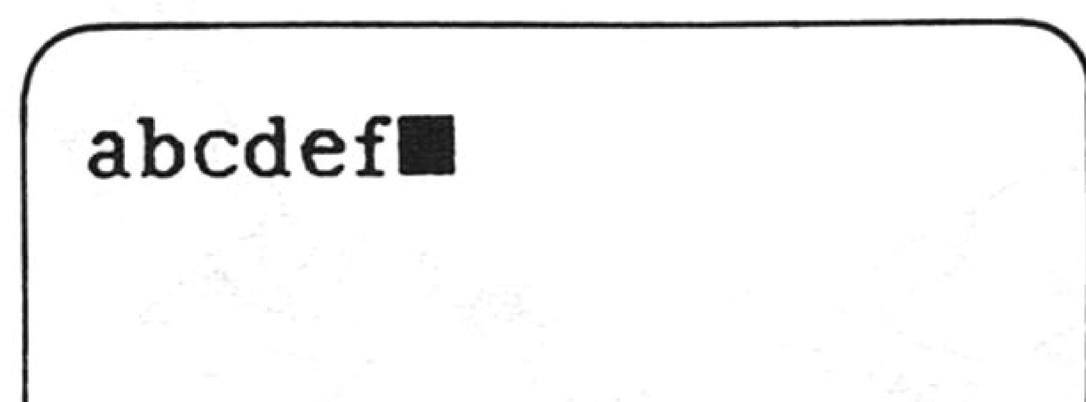


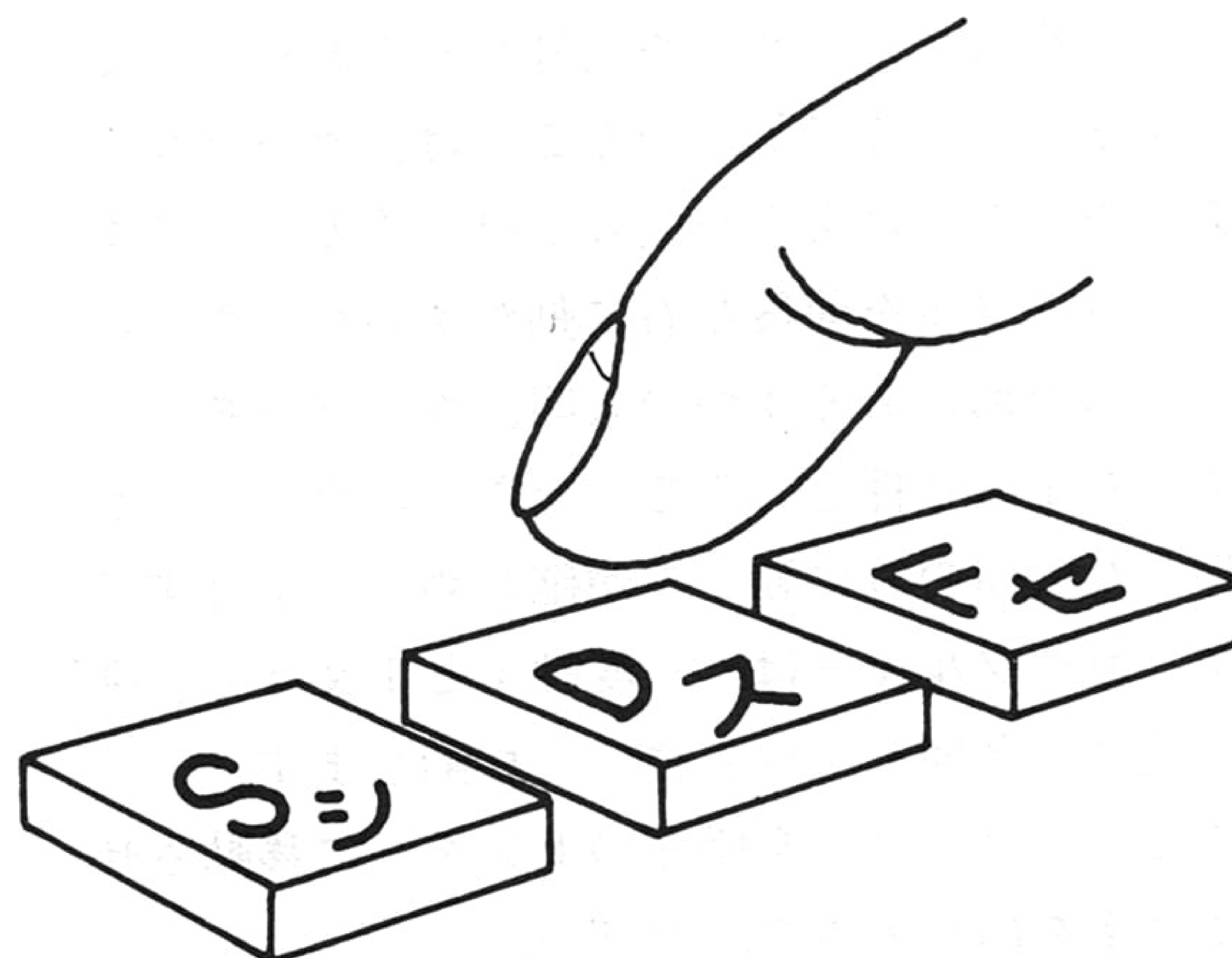
## 5-3 アルファベットの小文字／大文字の使いわけ

### (1)小文字主体の入力のとき

● **小文字の出し方**

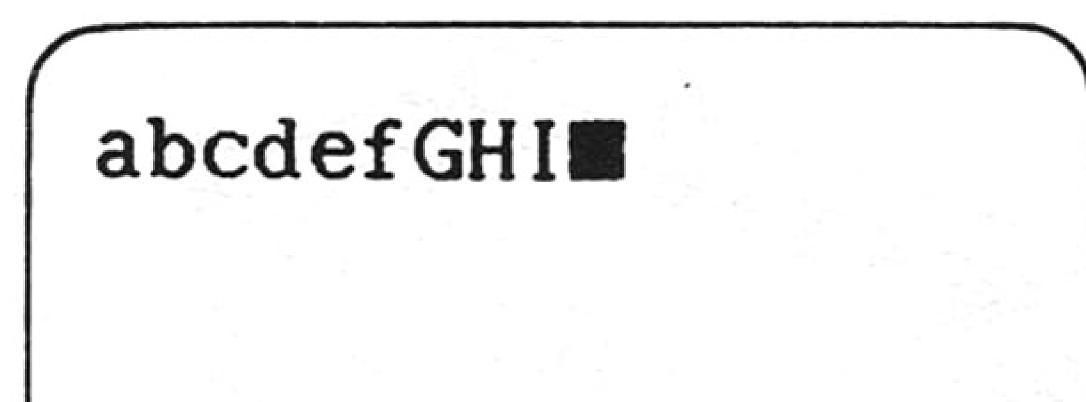
アルファベットのキーをそのまま押せば、カーソルのある位置に小文字が表示されます。

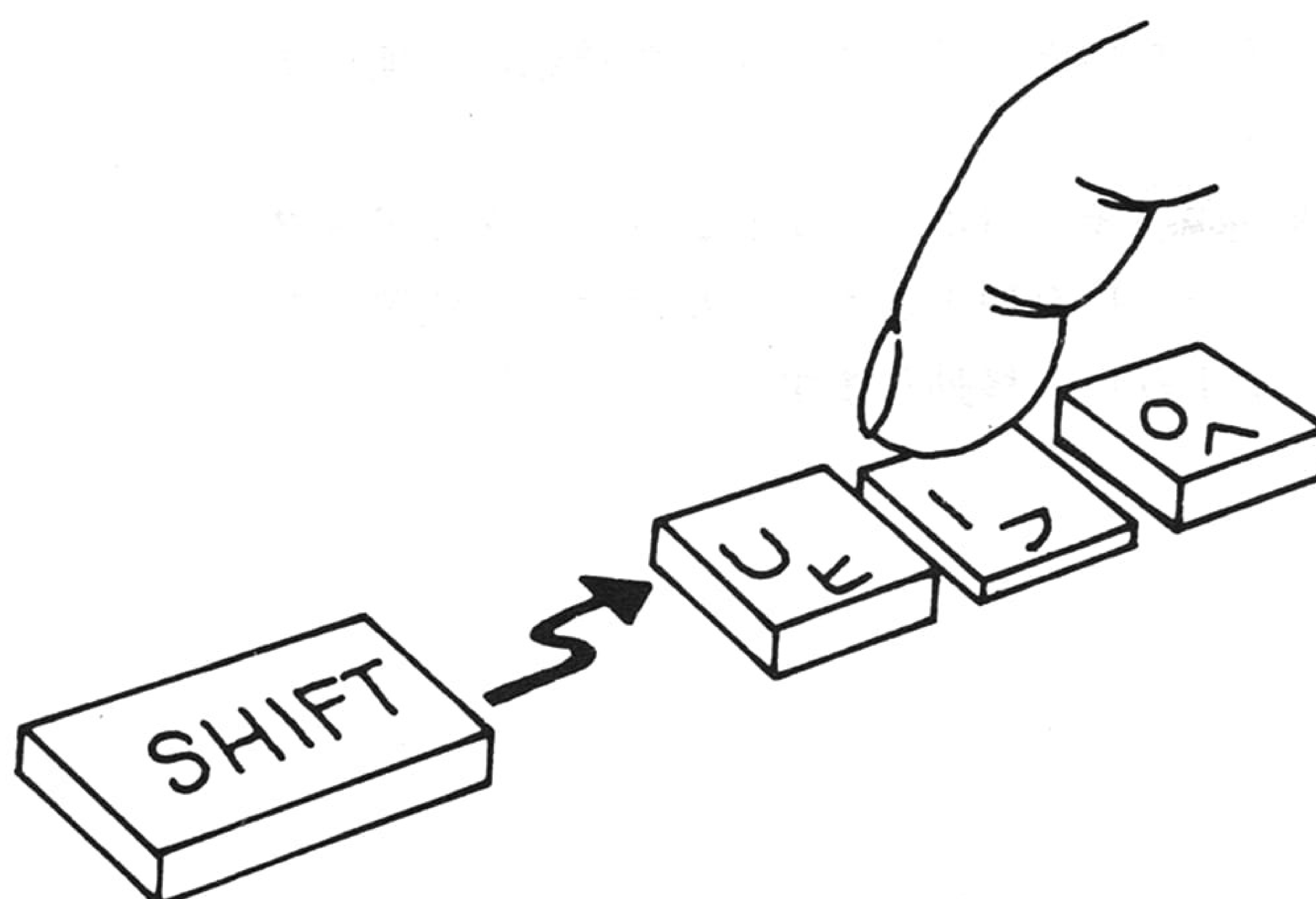



● **大文字の出し方**

SHIFT キーを押しながら、文字キーを押します。



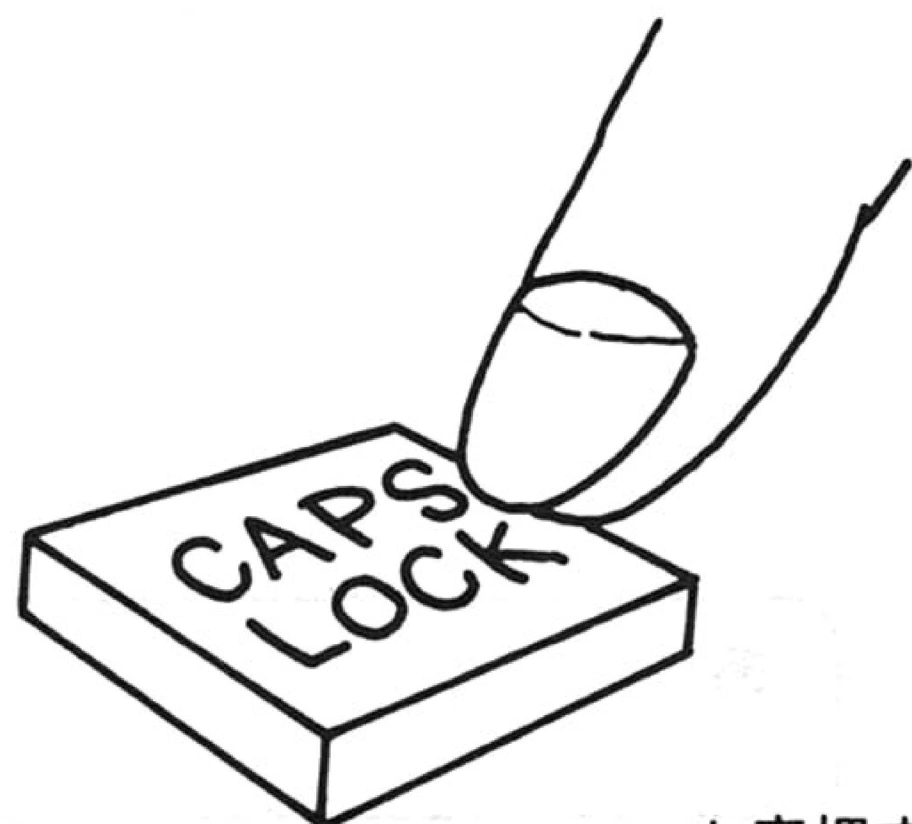




このように SHIFT キーを押したり離したりしながら、大文字と小文字を打ち分けることができます。ところが、大文字の多いプログラムを入力する場合には CAPS LOCK キーを使う方法が簡単です。

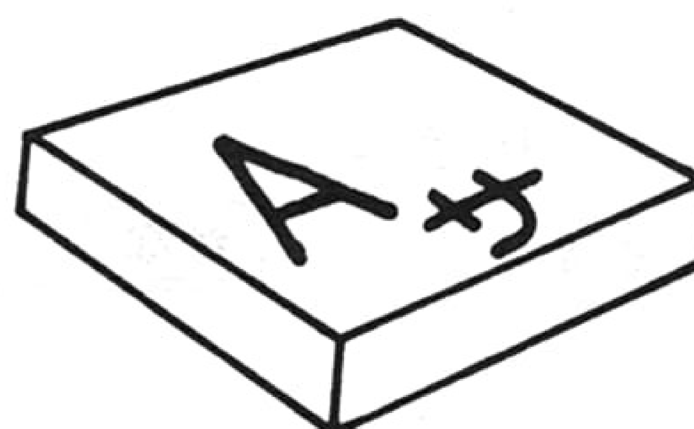
## (2)大文字主体の入力のとき

**●大文字の出し方**

CAPS LOCK キーを押してから文字キーを押します。




1度押すとロック状態になる




"A" が表示される

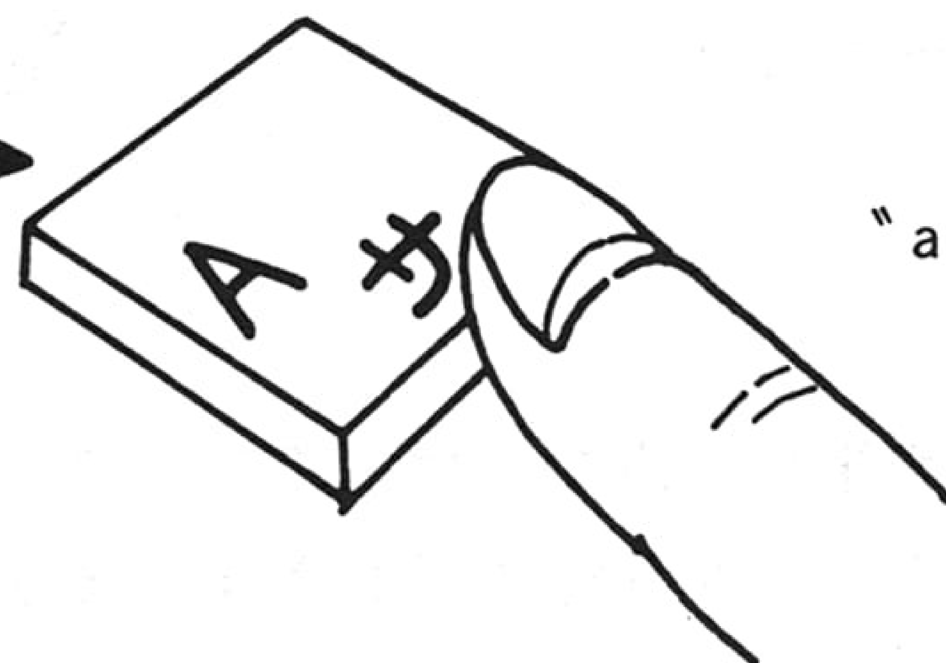
**●小文字の出し方**

CAPS LOCK キーをロック状態のまま SHIFT キーを押しながら文字キーを押します。




ロックしたまま




"a" が表示される

練習が終わったら CAPS LOCK キーをもう一度押して、ロックを解除してください。

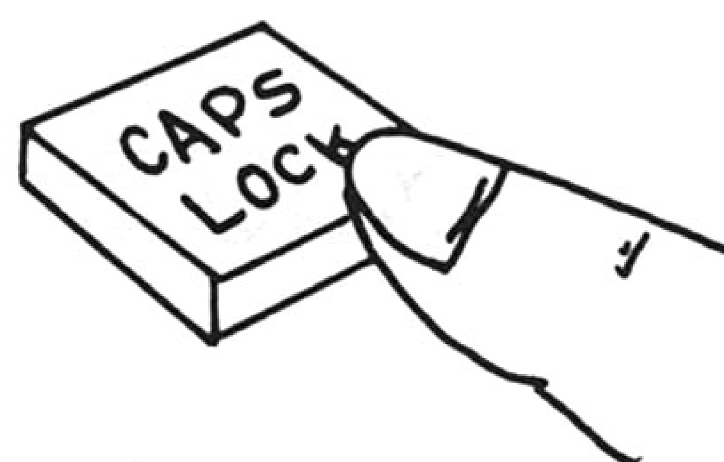
### CAPS LOCK キーの使い方

CAPS LOCK キーは押すごとにロック状態とその解除をくりかえします。打とうとする英文の中に小文字が多いか大文字が多いかで使いわけると便利です。




1度押すとロック状態となる。

ロック状態が続く




もう1度押すとロックは解除される

再び押すまで解除状態が続く

## 5-4 かなの出し方

キーの上にはアルファベットや記号といっしょにカタカナが書いてありますね。MX-101はカタカナはもちろんですが、ひらがなも出すことができます。これには[かな]キーを使い、この[かな]キーも[CAPS LOCK]キーと同じように、押すごとにロック状態と解除状態をくりかえしますから、押しつづける必要はありません。

### ●ひらがなを出すには

| | | |
|---|---|---|
| [かな]キーを押してロック状態にしてから文字キーを押します。 | [かな] [1 ! ア ァ] | あ |
| だく点をつけるには | [Y ハ] [: * ゛ -] | あは゛ |
| 半だく点をつけるには | [Y ハ] [] ! ゜ 」] | あは゛は゜ |
| 小文字を出すには | [SHIFT] → [1 ! ア ァ] | あは゛は゜ぁ |
| 音引きを出すには | [Y ハ] [SHIFT] → [: * ゛ -] | あは゛は゜ぁはー |

※だく点や半だく点などは、ダブルクォーテーション(")や句点(。)とまちがえやすいのでキーの位置をしっかり覚えておきましょう。

次は、カタカナの出し方の練習をしますから、[かな]キーはロックしたままにしておきます。

### ●カタカナの出し方

| | | |
|---|---|---|
| [かな]キーがロックされている状態で[CAPS LOCK]キーを押して文字キーを押します。 | [かな] [CAPS LOCK] [1 ! ア ァ] | ア |
| だく点をつけるには | [Y ハ] [: * ゛ -] | アハ゛ |
| 半だく点をつけるには | [Y ハ] [] ! ゜ 」] | アハ゛ハ゜ |
| 小文字を出すには | [SHIFT] → [1 ! ア ァ] | アハ゛ハ゜ァ |
| 音引きを出すには | [Y ハ] [SHIFT] → [: * ゛ -] | アハ゛ハ゜ァハー |

### ひとつのキーをいく通りにも使えるシフトキー

大文字、小文字などを出すのに活躍した SHIFT キーは、ひとつのキーをいく通りにも使うためにあるキーです。
右図のように、ひとつのキーには最大４つの文字や記号が書かれていますが、このキーを押しますと〝2〟が表示されます。
ところが SHIFT キーを押しながら、このキーを押すと、上の " (ダブルクォーテーション)が出ます。 このように、 キーの上側にある記号を出すには、SHIFT キーを押しながら使うことを覚えてください。
次に、かな 、CAPS LOCK などがロックされている状態のまま SHIFT キーを使うと右側の文字または記号が表示されます。




## 5-5 文字、記号の出し方のまとめ

これで、いろいろな文字や記号が出せるようになったはずですが、ここでいままでのことをまとめておきましょう。

| | ロック状態 | 文字の種類と位置 | 表示 | SHIFT を押しながら | 表示 |
|---|---|---|---|---|---|
| (例１) | 何も押さない | アルファベットの小文字／キーの左 | a | アルファベット大文字／キーの左 | A |
| Aサ | CAPS LOCK (ロック) | アルファベットの大文字／キーの左 | A | アルファベット小文字／キーの左 | a |
| | かな (ロック) | ひらがな／キーの下 | さ | ――― | ― |
| | かな CAPS LOCK (ロック) | カタカナ／キーの下 | サ | ――― | ― |
| (例２) | 何も押さない | 数字／キーの左 | 4 | 特殊文字／キーの上 | $ |
| 4 $ エ ェ | かな (ロック) | ひらがな大文字／キーの下 | え | ひらがな小文字／キーの右 | ぇ |
| | かな CAPS LOCK (ロック) | カタカナ大文字／キーの下 | エ | カタカナ小文字／キーの右 | ェ |
| | GRAPH 押しながら | グラフィックパターン | 水 | ――― | ― |

## 5-6 記号が似ているから注意しましょう

キーボードには、似たような記号があります。ひとつひとつ意味がちがいますから、しっかり覚えてください。

| 記号／キーマーク | ― | 記号／キーマーク | " | 記号／キーマーク | ∧と^ |
|---|---|---|---|---|---|
| [ー ＝ ラ] | 数字のマイナス記号<br>英語のハイフォン | [2 " イ ィ] | ダブルクォーテーション | [^ ~ リ] | 数字のベキ乗マーク |
| [: * ー] | ひらがな、カタカナの音引き | | | | |
| [＿ ン ・] | アンダーバー | [: * ゛] | ひらがな、カタカナのだく点 | [0 ^] | ひらがな、カタカナのへ |

| 記号／キーマーク | ○ | 記号／キーマーク | ● | 記号／キーマーク | ' |
|---|---|---|---|---|---|
| [] 」 ゜] | ひらがな、カタカナの半だく点 | [＿ ン ・] | 中黒点 | [7 ' ニ] | リマーク<br>(アポストロフィー) |
| [/ ? ヲ 。] | 句　点 | [. > ワ 、] | ピリオド | [, < ヨ] | カンマ |

| 記号／キーマーク | < >と( ) |
|---|---|
| [, < ヨ] | 不 等 号 |
| [8 ( ヌ] | カ ッ コ |

## 5-7 スペースバーの働き

これで、大文字、小文字や数字が自由自在に出せるようになったはずです。では、ここで問題を出します。次の文を画面に表示してみてください。

We call the apple "FUJI"

さて、Weとcallの間に1文字分あいていますが、これはどうしますか？

「カーソルを1文字分移動すればいいのでは」──正解です。それで十分ですが、もうひとつ方法があります。それは、1字あけるところにきたら、キーボードの下の方にあるスペースバーを押すのです。では、スペースバーの働きは、カーソルキーの[→]の働きとまったく同じかというと、そうではありません。それを確かめてみます。

①まずカーソルキー[←]を使って、先ほど表示した文の先頭のWのところまでカーソルを移動してください。

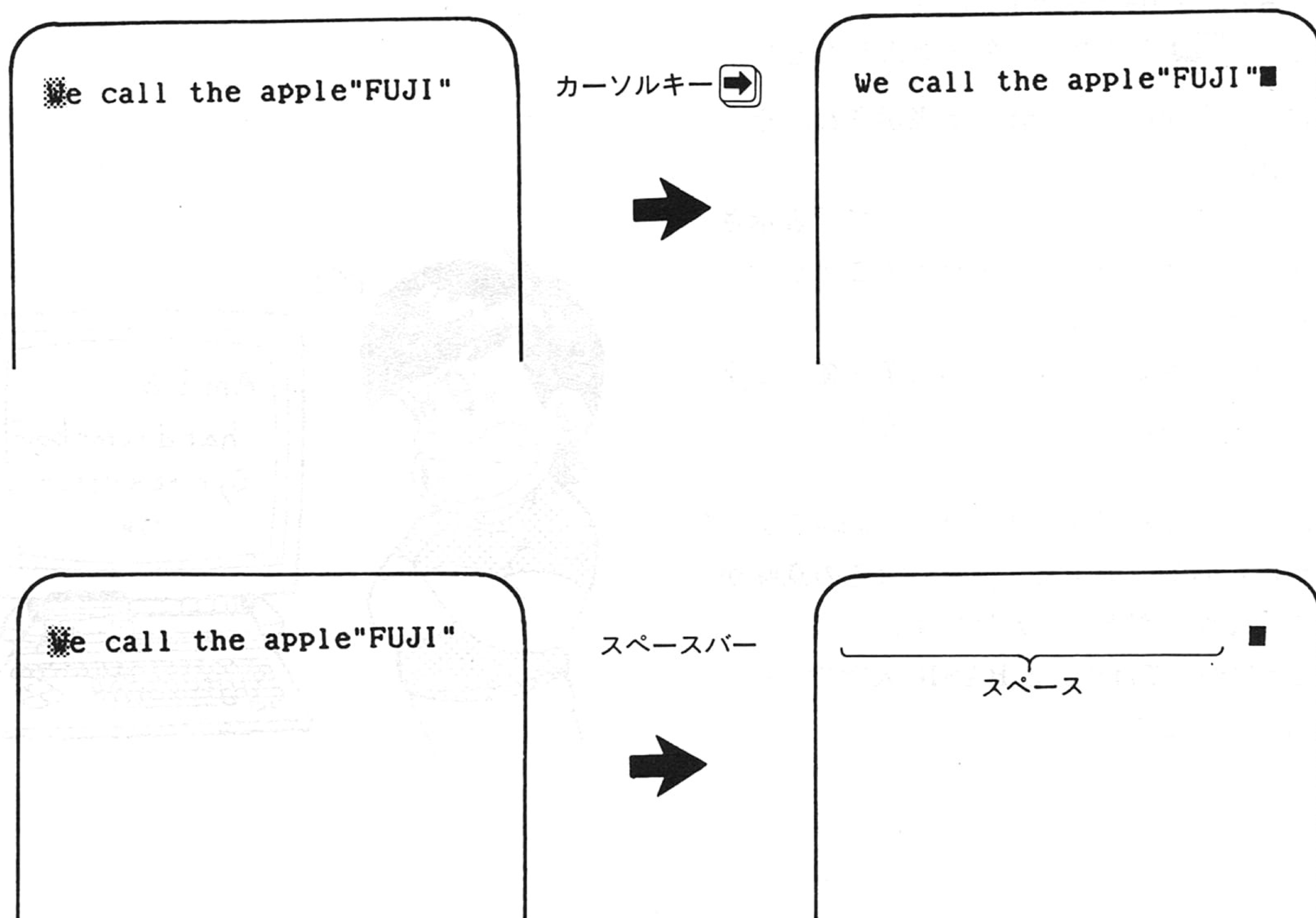
次にカーソルキー[→]を最後の " まで押し続けてください。

②同様に再びカーソルをWのところまで移動します。
　こんどはスペースバーを押し続けてください。

①と②では結果がちがいましたね。②ではせっかく出した文字が消えたはずです。カーソルキーとスペースバーのそれぞれの役割は次のようになっているのです。

・カーソルキー ➡
　……カーソルを右に1文字分移動させる
・スペースバー
　……スペース（空白）を1つ出す。

したがって、もともと何も表示されていないところにカーソルがある場合は、2つのキーは全く同じ働きをしますが、カーソルの下に文字がある場合は、単にカーソルを移動させるのと空白を出すのとではちがった結果になるわけです。



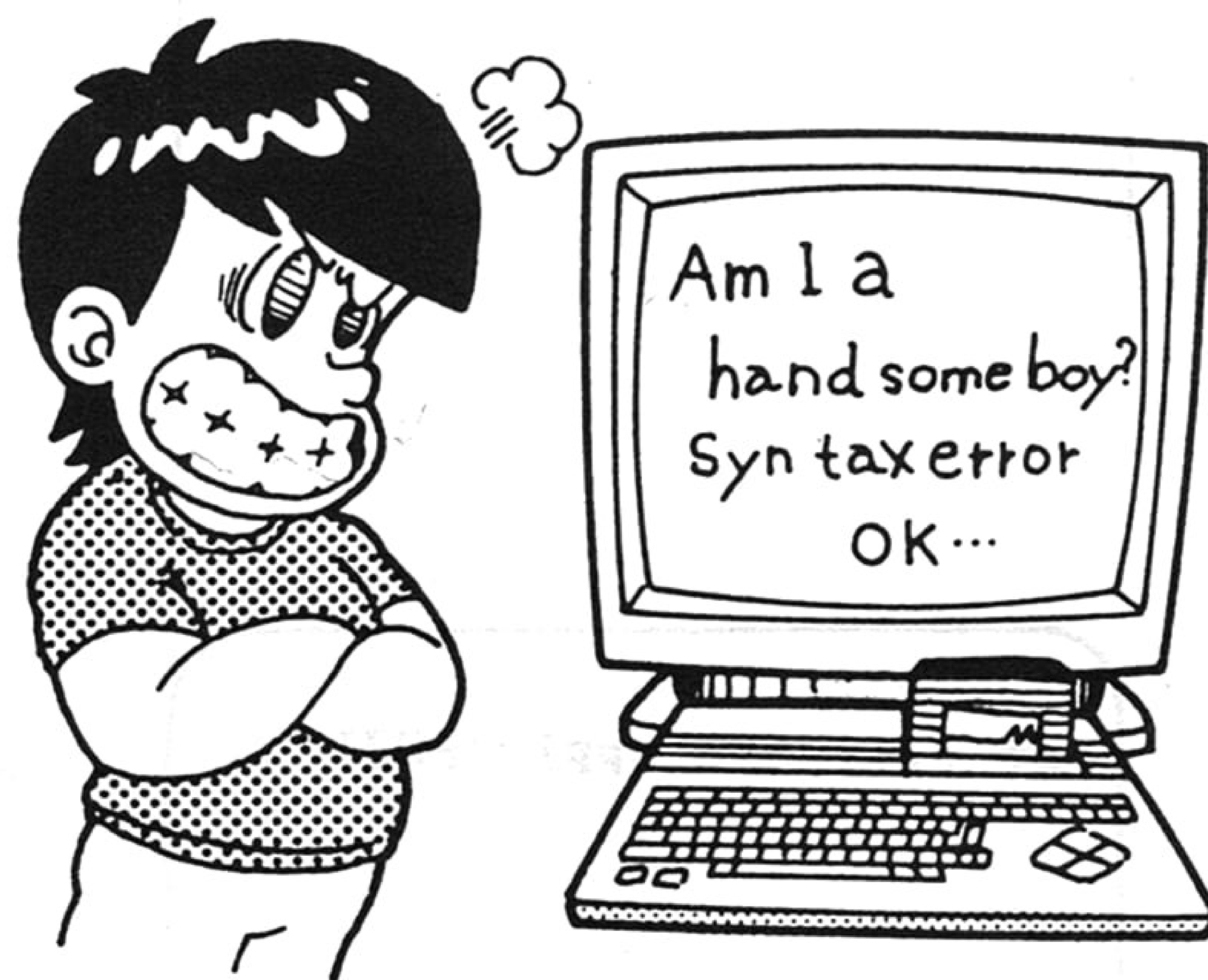

コンピュータの本では、スペースを出すことを〝␣〟で表わしてあるものがありますが、これは〝␣〟1つで、スペース1コという意味です。

## 5-8 RETURNキーについて

これまで画面に文字を表示してきましたが、コンピュータの反応はありませんでした。いよいよこれからコンピュータの性能を引き出すことにしましょう。
キーボードの右側にあるリターンキー ↵ を見つけてください。矢印が変形したような絵が書いてあるこのキーがとても大切なキーなのです。
いままで画面に表示された文字は、これはあくまでも〝表示〟であって、コンピュータはまだ読みとっていません。ですからコンピュータが反応しないのは当然です。では、コンピュータが表示した文字を読みとるにはどうしたらいいでしょうか？
ここで活躍するのがリターンキーです。
このキーは、コンピュータに対して、「画面の文を読みとってください」と命令するもので、これが押されることによって、はじめてコンピュータは画面に書かれた文字を読みとるのです。
そこで、たとえば

Am I a handsome boy ?

と表示し、↵ を押して答を要求したとしましょう……。
画面には〝Syntax error〟と表示され、その下に〝Ok〟と書かれたはずです。
これはシンタックス・エラーといって、「表示された文がコンピュータには理解することができません」という答えなのです。
私たちが、日本語という特定の言葉を使って会話をするように、コンピュータにもコンピュータが理解できる言葉があります。
今回は、いきなり〝Am I a handsome boy ?〟と聞かれたので、コンピュータにはその意味が理解できなかったというわけです。
このことについては第2部BASIC入門でしっかり学ぶことにしましょう。



## 5-9 ファンクションキーの働き

キーボードの上の方に [F1 F6] ～ [F5 F10] というキーがありますが、これをファンクションキーといいます。

ファンクションキーは、比較的よく使われるコマンド（コンピュータに命令することば）を1文字ずつ入力しなくても、一度の操作で入力できる便利なキーです。

F1からF5まではそのまま押し、F6からF10までは [SHIFT] キーを押しながら押してコマンドを表示させます。

なお、このコマンドは自由に設定することができます。そして、そのときはKeyという命令で変更します。(136P参照)

| キー | 文字キーでいれたときの働き（設定されているコマンド） | | 文字キーでいれたときの働き（設定されているコマンド） |
|---|---|---|---|
| [F1] | [C][O][L][O][R][スペース] | [F6] | [C][O][L][O][R][スペース][1][5][,][4][,][7][↵] |
| [F2] | [A][U][T][O][スペース] | [F7] | [C][L][O][A][D]["] |
| [F3] | [G][O][T][O][スペース] | [F8] | [C][O][N][T][↵] |
| [F4] | [L][I][S][T][スペース] | [F9] | [L][I][S][T][.][↵] |
| [F5] | [R][U][N][スペース][↵] | [F10] | [C][L][S][:][R][U][N][↵] |

※画面の下に color auto goto list run と表示されていますが、これは [F1] に color が、[F2] に auto が設定されていますという、ファンクションキーの働きを表示しているコマンドガイドなのです。[SHIFT] キーを押せば [F6] から [F10] までの内容を表示します。

※このコマンドガイドを消したいときは [K][E][Y][O][F][F][↵] と操作してください。
再び表示させるときは [K][E][Y][O][N][↵] と操作します。

## 5-10 特殊キー

これまで何度も使ってきた SHIFT キーや、かな キーなどや文字キー、ファンクションキー以外の特殊キーの働きをここでまとめておきましょう。

| 特殊キー | 読み方 | 働き |
|---|---|---|
| TAB | タブキー | カーソルを8文字分移動させます。<br>（その間に文字がある場合、文字は消えてしまいます） |
| SHIFT | シフトキー | 文字キーと組み合わせて、アルファベットの文字や、キーの上と右に書かれている文字や記号を表示します。 |
| CAPS LOCK | キャプスロックキー | アルファベット大文字、かな キーと組んでカタカナを表示します。 |
| | スペースバー | 1文字分の空間を表示します。<br>（文中では スペース と記します） |
| GRAPH | グラフィックキー | 文字キーと組み合わせてグラフィックパターン(29P)を表示します。 |
| かな | かなキー | 文字キーと組み合わせて、ひらがな、カタカナを表示します。 |
| ↵ | リターンキー | 入力した文字や記号をコンピュータに記憶させると同時にカーソルを次の行の先頭に移します。 |
| BS | バックスペースキー | カーソルを1文字前に戻し、そこにある文字を消します。 |
| ↑ ↓ → ← | カーソルキー | カーソルをそれぞれ矢印の方向に1文字分移動します。 |
| CLS HOME | ホームポジションキー<br>クリアスクリーンキー | カーソルを画面の左上端に移動します。<br>SHIFT キーと組み合わせて、画面の文字をすべて消し、カーソルを画面の左上端に移動します。 |
| INS | インサートキー | 文字と文字の間に、別の文字を入れます。(詳しいことは、第2部〝BASIC入門〞で説明します。) |
| DEL | デリートキー | カーソル位置にある文字を消します。(同上) |
| STOP | ストップキー | プログラムの実行や、リストの表示を一時停止します。もう一度押すと再び続きがはじまります。 |

※ ESC (エスケープキー)、SELECT (セレクトキー)、TR1 TR2 (トリガーキー)、ジョイパッドはプログラム入力の段階では使いませんので、ここでは説明を省略します。

※ CTRL キーは、他のキーと組み合わせて、下記のようなさまざまな働きをします。

| キー | CTRL といっしょに押したときの働き |
| --- | --- |
| B | カーソルを直前の単語の先頭へ移動します。 |
| C | 入力待ちの状態を中断します。 |
| E | カーソルのある行のカーソル以下の文字を消します。 |
| F | カーソルを次の単語の先頭へ移動します。 |
| G | BEEP音を出します。 |
| H | カーソルの1つ前の文字を消す。(BS と同じ) |
| I | カーソルを8文字分移動する。(TAB と同じ) |
| J | カーソルを次の行に移動する。 |
| K | カーソルを画面の左上端に移動する。(CLS HOME と同じ) |
| L | 画面の文字をすべて消し、カーソルを左上端に移動する。<br>(SHIFT → CLS HOME と同じ) |
| M | カーソルを左端に戻します。(↵ と同じ) |
| N | カーソルを行末に移動します。 |
| R | インサートモードにします。(INS と同じ) |
| U | カーソルのある行をすべて消します。 |
| ¥ | カーソルを右に移動します。(→ と同じ) |
| ] | カーソルを左に移動します。(← と同じ) |
| ^ | カーソルを上に移動します。(↑ と同じ) |
| — | カーソルを下に移動します。(↓ と同じ) ※ — は SHIFT → シ と押します。 |
| STOP | プログラムの実行やリストの表示を中断します。<br>リスト表示はこれで終了し、プログラム実行を終了せず再実行する場合は C O N T ↵ と操作します。 |

# 第6章 テープソフトを使ってみましょう

## 6-1 テープソフトとは

第4章で、ゲームソフトにはROMカートリッジとテープがあるといいましたが、ここではそのテープについて説明します。
ゲームなどのプログラムの入ったテープのことをテープソフトといい、外見は普通のカセットテープです。使い方はROMカートリッジに比べると、多少手間がかかりますが、一度覚えてしまえば簡単ですからがんばってチャレンジしてください。

## 6-2 ゲームの準備

テープレコーダーが､カセットインタフェイスでパソコン体とつながれていることを確かめてから､次の手順で操作します。(リモート端子のないテープレコーダーについては、次のページで説明します。)

①テープをテープレコーダーにセットします。テープのA面とB面でちがったゲームのプログラムが入っていることがあるので、注意してください。

②リモートプラグをはずし､『巻きもどしボタン』を押し、巻きもどしが終ったら、リモートプラグを差し込みます。(カシオKR-10はリモートプラグをはずす必要はありません)

③『再生ボタン』を押します。(このとき、テープはまだ回っていません)

④これでテープの方の準備ができましたので、次にキーボードから次のようにキーを押してください。

C L O A D ↵

CLOADは､『シーロード』と読み、CASSETTE LOADの略で、カセットテープの内容をパソコンの記憶部分（メモリー）に読み込むことです。
また、テープの片面に何種類かのゲームプログラムが入っている場合は､CLOADのあとに､ゲーム名（ファイルネーム）をつけることもあります。(詳しくは『第2部BASIC入門』の45Pを参照)

⑤まもなく画面は下のようになります。これは、パソコンがテープのなかから×××という名前のプログラムを見つけたという意味です。見つけたプログラムをパソコンのなかに、ロードし終わると"Ok"と表示されてテープが止まり、これで完了です。

```
cload
Found:xxx
```

←×××というファイルネームのプログラムを見つけた。

## 6-3　最後の操作

画面に〝Ok〞が表示されたら、キーボードで次のように入れてください。

R U N ↵

RUNというのは、パソコンに対して「プログラム通り仕事を始めなさい」という命令です。ですから、プログラムを始めることを〝RUNさせる〞とか〝実行させる〞とか〝走らせる〞といいます。
やがて画面がパッと変って、ゲームが始まります。あとはROMカートリッジの時と同様、画面の説明およびテープソフトの入っていた箱の裏面や説明書を読んでゲームを楽しんでください。

## 6-4　テープレコーダーにリモート端子がないときは……

家庭用テープレコーダーのなかには、リモート端子がついていないものもあります。そういう機種のときは次の操作手順で行なってください。
①テープをセットし、巻きもどして準備を完了させます。
②リモート端子以外の接続を確かめたら、キーボードでCLOAD ↵ を入れます。

C L O A D ↵

③テープレコーダ ーの再生ボタンを押します。(テープが回り始めます)
④あとはリモート端子付のときと同じです。〝Ok〞の表示がでたら、テープレコーダーを停止させてください。
この後の操作は、リモート端子付と同様、 R U N ↵ でゲームを開始してください。

## 6-5　アレッ、困ったなァ…　というときは

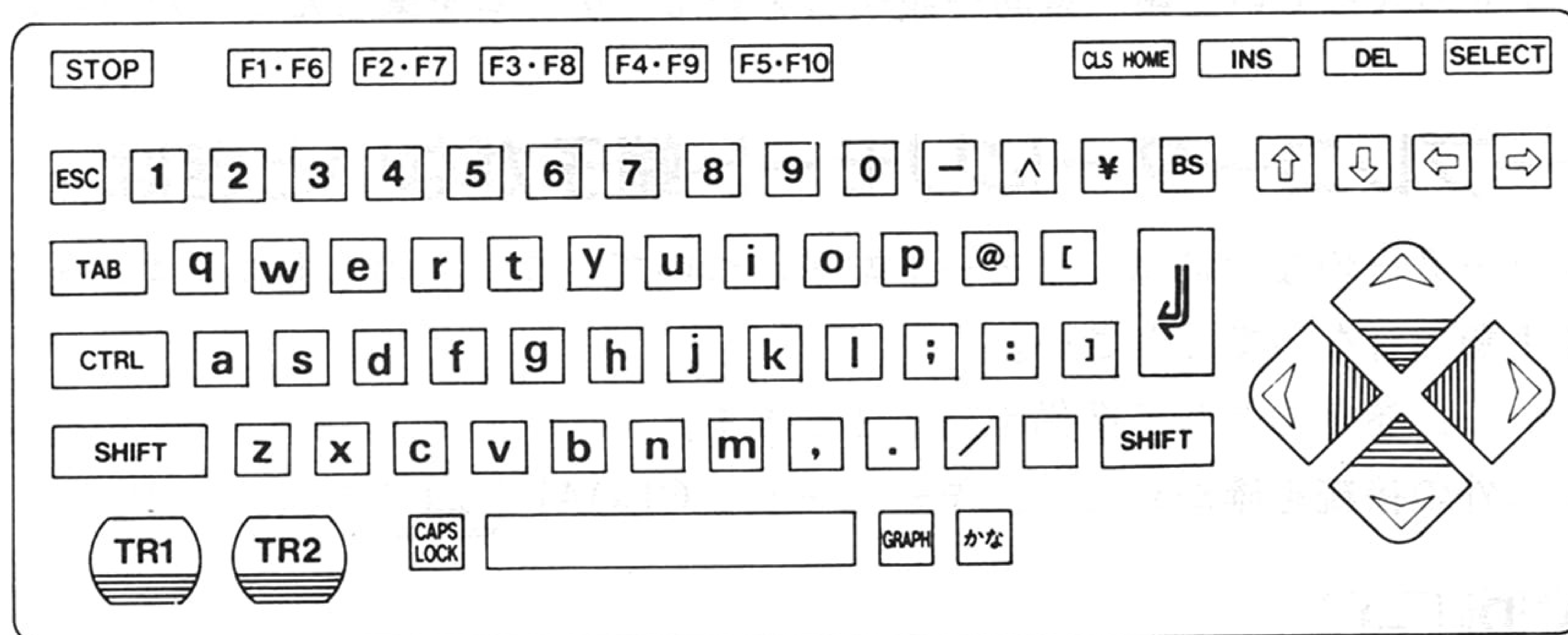
①テープが回っているのに〝Found〞が出ない。
キーボード左側の CTRL キーを押しながら左上の STOP キーを押すと〝Device I/O error〞が画面に表示され、テープは止まります。そこでもう一度テープを巻きもどし、テープレコーダーの音量(ボリューム)を大きくしたり、音質(トーン)を高くして、最初からやり直してください。(音量を最大にしても同じ場合は、他のテープレコーダーで試してください。)
②〝Ok〞が出る前に〝Device I/O error〞が出てテープが止まる
原因としては、テープに傷がついているか、テープレコーダーの電源が途中で切れたことなどが考えられます。したがって、この2つのことを確かめてからもう一度やり直してください。
テープレコーダーのヘッドを市販のクリーニングキットできれいにすれば、直ることもあります。

## ●まとめ————いろいろなキー機能

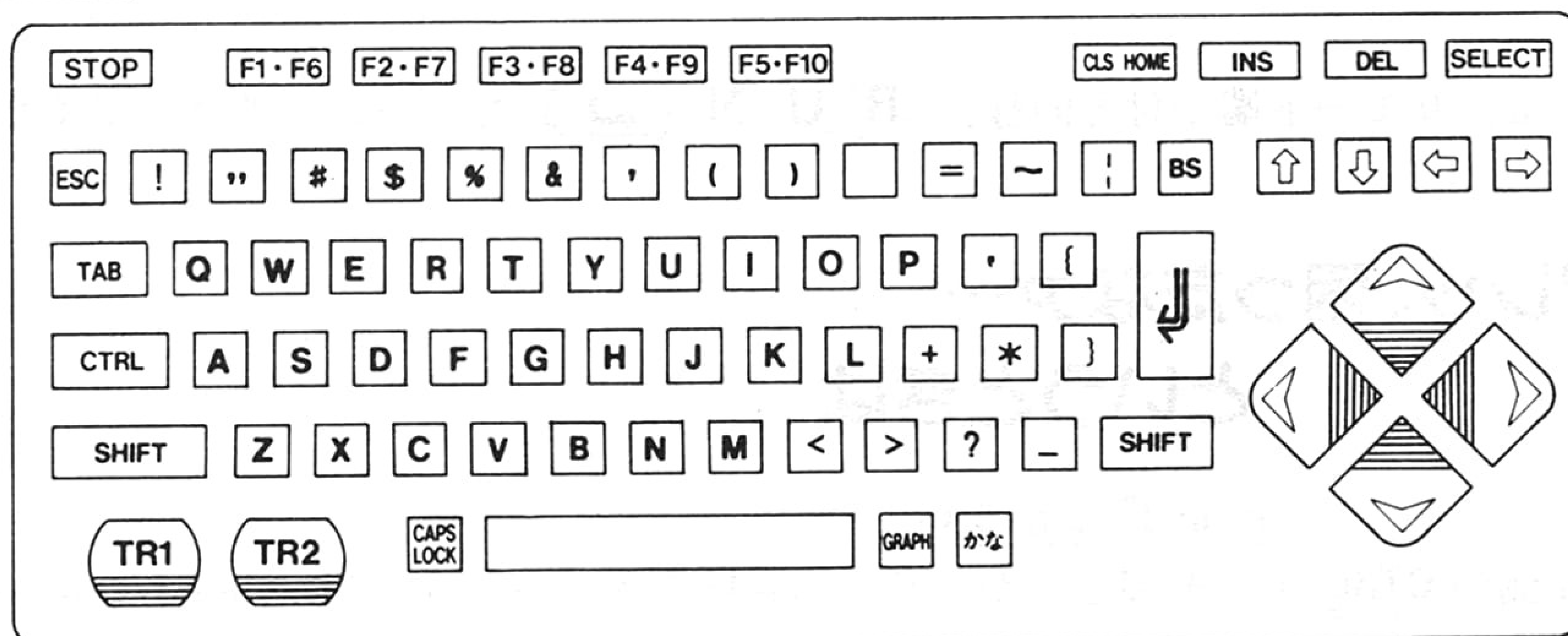
特殊キーと文字キーを組み合わせて押すと、いろいろな文字や記号が表示されることがわかったと思います。次に特殊キーを押したときの各キーの機能をまとめておきますので、プログラムを入力するとき、どのキーを押せばよいかわからないときなどに利用してください。

### アルファベット

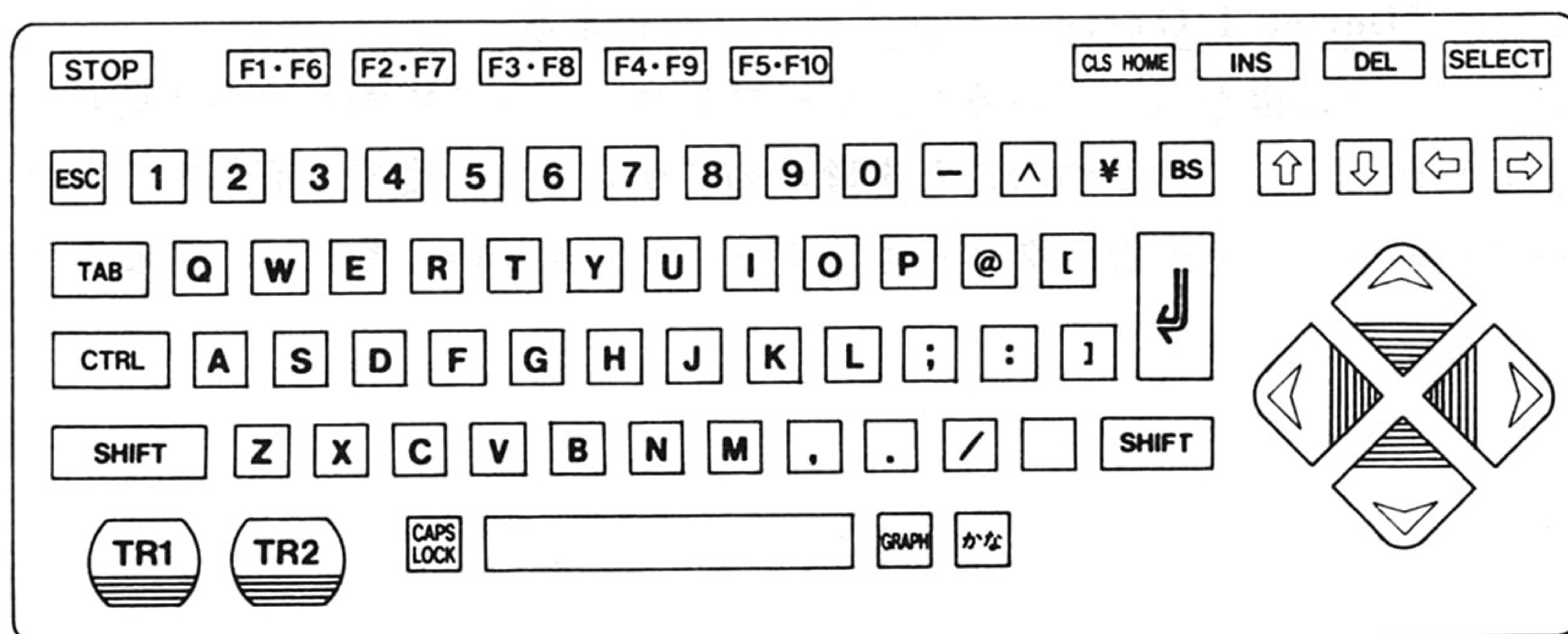
(文字キーだけを押したとき)




( SHIFT キーを押しながらのとき)

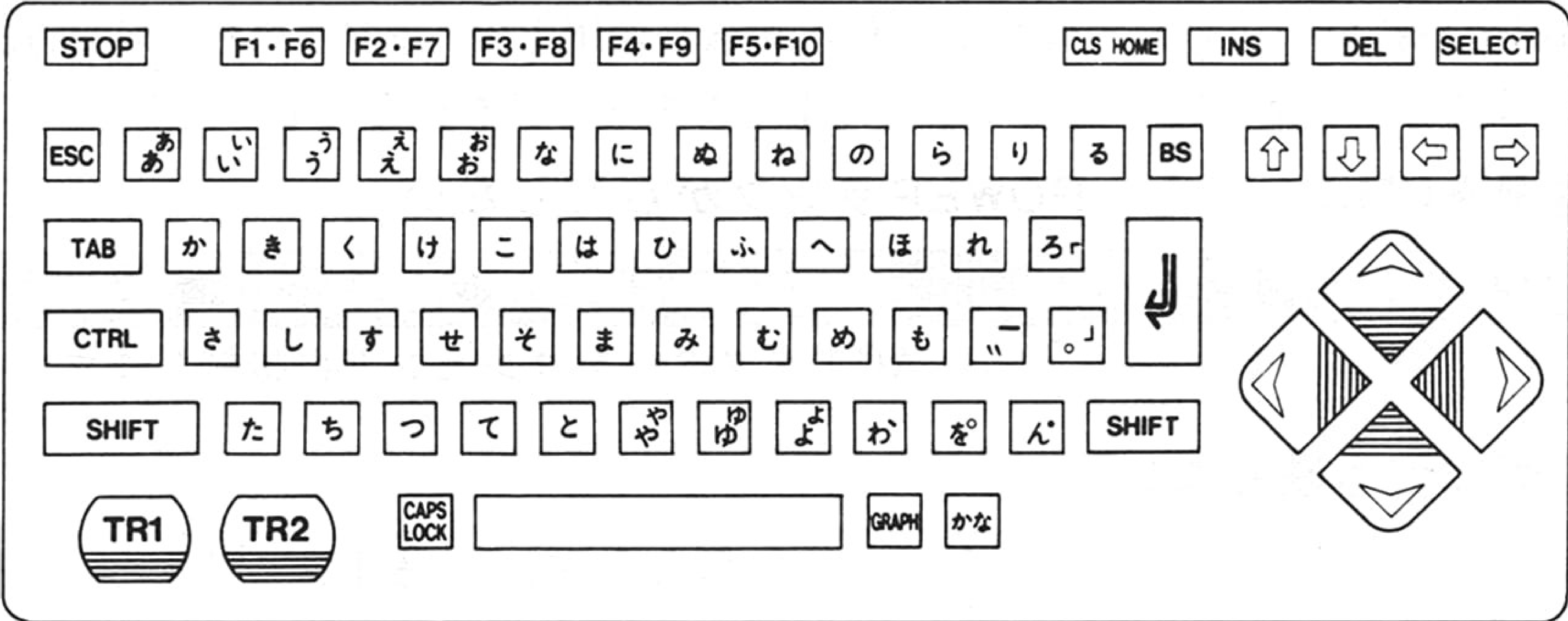



( CAPS LOCK キーが押してあるとき)



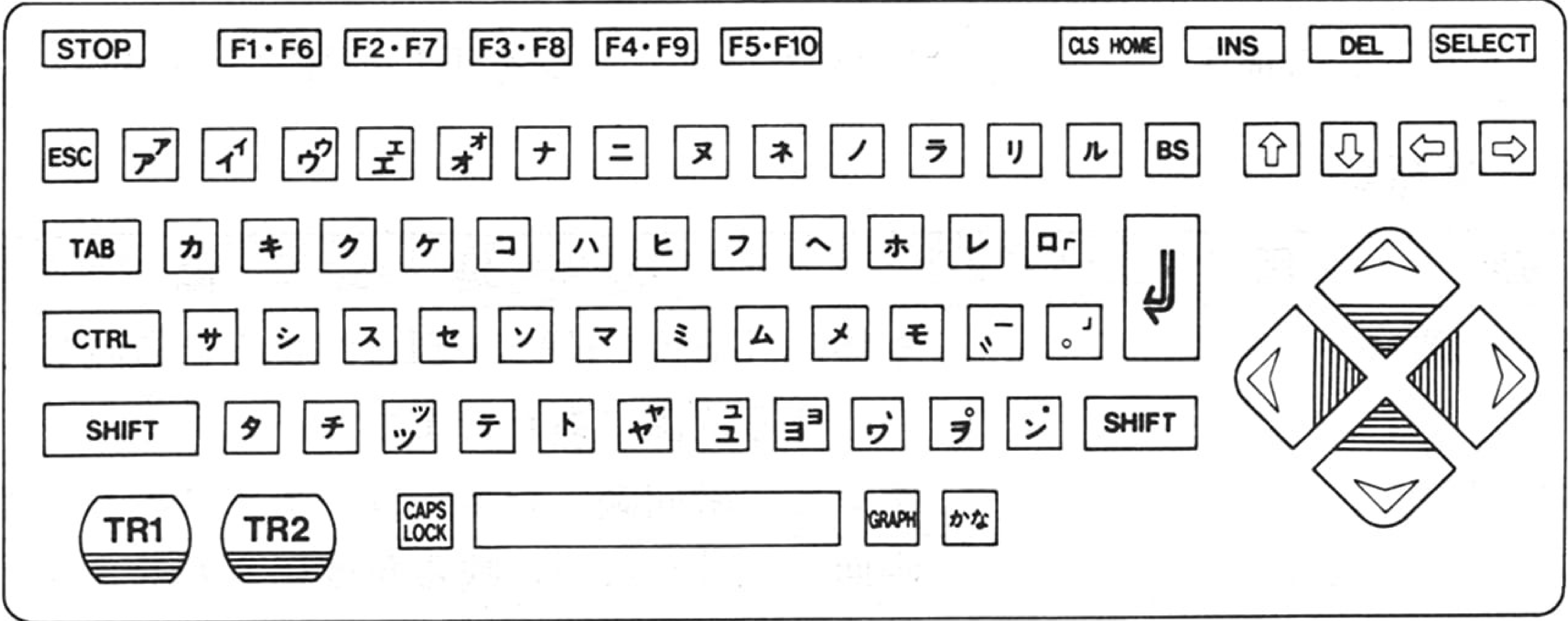
## ひらがな

（かな キーが押してあるとき）



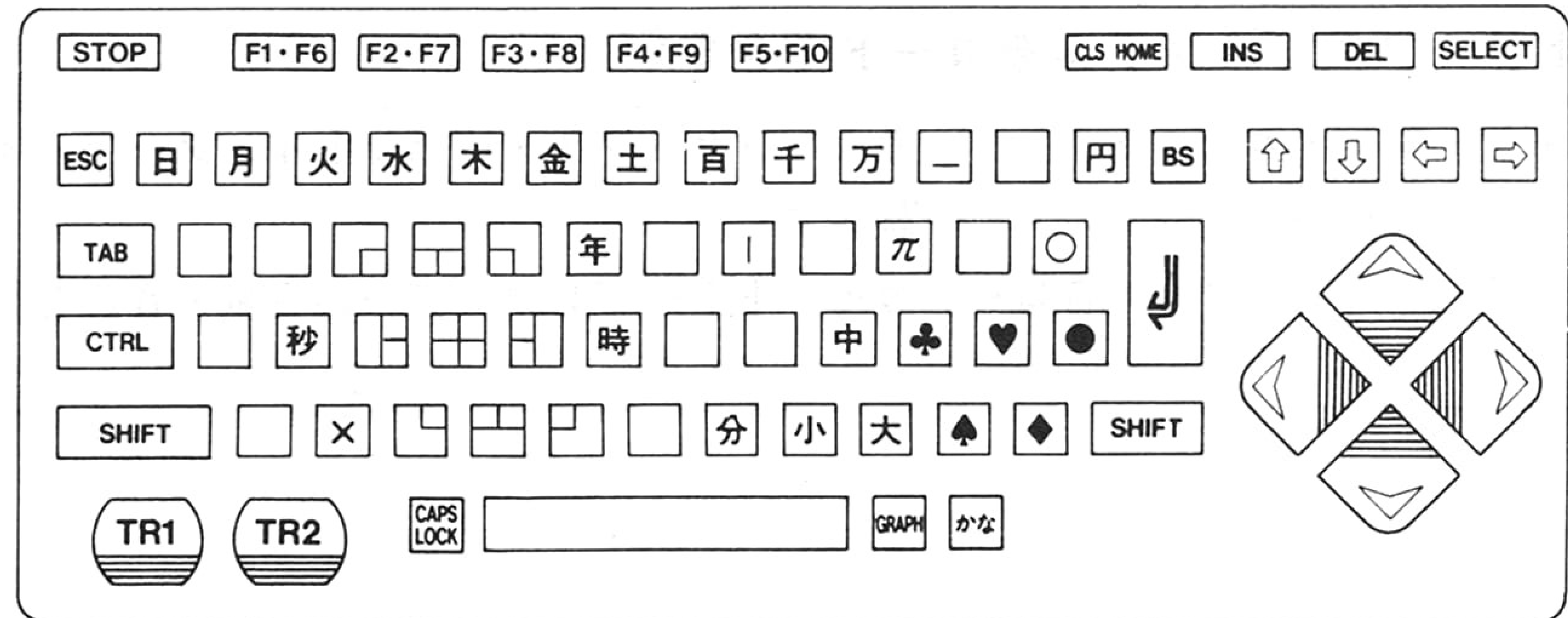

## カタカナ

（かな キーと CAPS LOCK キーが押してあるとき）




## グラフィック

（GRAPH キーを押しながらのとき）



# 第7章 トラブルシューティング

以上でMX-101の取扱い方法の説明は終りですが、最後に使用中のトラブルとその対処方法について述べておきます。故障とお考えになる前に、以下のようなチェックを行なってください。

### ●本体の電源をONにしても、POWERランプが点灯しない

| 原　　因 | 処　　置 |
|---|---|
| ACアダプター | ACアダプターをコンセントに接続します。 |

### ●テレビ画面に何も映らない

| | |
|---|---|
| 1.テレビの電源 | テレビの電源をONにします。 |
| 2.アンテナの接続 | 送信アンテナ、受信アンテナをそれぞれ正しく接続します。 |
| 3.「UHF CHANNEL」つまみの調整 | テレビ側で設定したチャンネル(13または14チャンネル)と同じ周波数になるように、つまみを微調整します。 |
| 4.テレビのチャンネルの周波数のずれ | テレビの取扱説明書にしたがい、チャンネルの周波数を正しく調整します。 |
| 5.ROMカートリッジの挿入 | ROMカートリッジを正しく挿入します。 |

### ●画像の映りが悪い

| 原　　因 | 処　　置 |
|---|---|
| 1.MX-101の微調整 | 本体の「UHF CHANNEL」つまみを微調整して一番きれいに映る位置にあわせます。 |
| 2.テレビの調整 | テレビの微同調、輝度、コントラストを調整します。 |
| 3.受信アンテナの角度調整 | 画面が一番きれいに映る角度にあわせます。 |
| 4.本体の送信アンテナと受信アンテナの間に障害物がある。 | 障害物を取り除きます。 |

### ●ROMカートリッジが作動しない

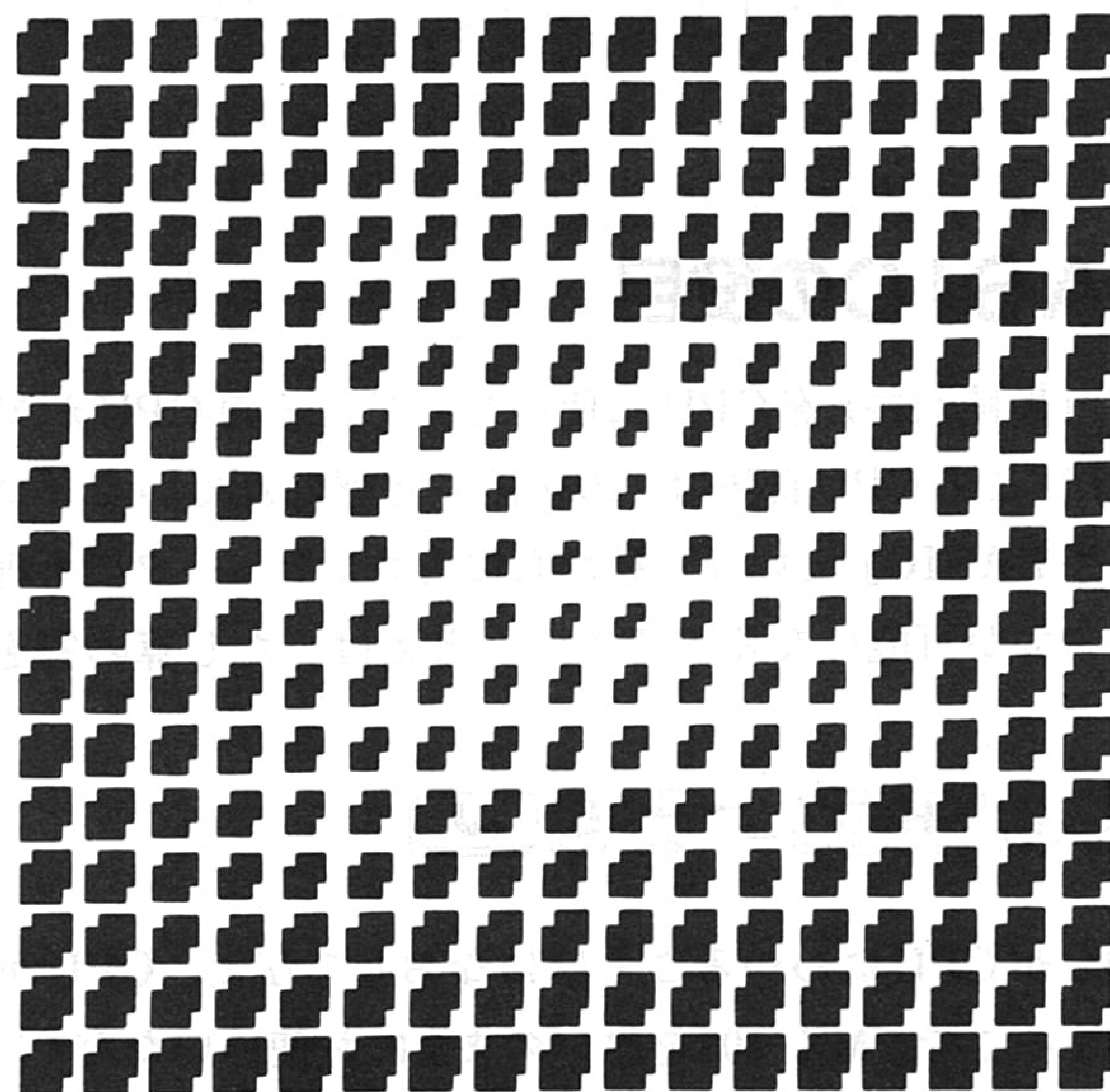
| 原　　因 | 処　　置 |
|---|---|
| 1.ROMカートリッジの挿入 | カートリッジを正しく入れます。 |

### ●カセットテープのプログラムをロードできない

| | |
|---|---|
| 1.カセットケーブル・カセットインターフェイスの接続<br>2.カセットテープレコーダーの音量・音質 | カセットケーブル・カセットインターフェイスを正しく接続する。<br>カセットテープレコーダーの音量・音質を調整します。 |

# 第2部

# BASIC入門



# 第1章 プログラムとは

## 1-1 コンピュータとは

ひと昔前まで、コンピュータに対する私たちのイメージは、〝人間が命令すれば何でもやってしまう〟スーパーマン的なものでしたが、現在では日常生活のなかにいろいろな形で組み込まれてきています。たとえば、お母さんなら「マイコン炊飯器」や「マイコン内蔵の洗たく器」がすぐに頭に浮かぶことでしょう。またビジネスマンなら「給与明細」や各種の「伝票」をイメージするでしょうし、子供たちはやはり「ＴＶゲーム」や「マイコン内蔵ラジカセ」などでしょうか。
このように、コンピュータはどんどん私たちの身近なものになりつつありますが、さてコンピュータとは何か？——ひと言でいえば「プログラムという命令によって動作するマシン」であり、オフィスコンピュータもパーソナルコンピュータもＴＶゲームも、すべてプログラムによって動いているのです。

## 1-2 MSX BASICの役目

MX-101は、Z-80というIC（集積回路）をCPUに使っていて、そのCPUがMX-101の中心となり、MX-101を制御する部分です。このCPU自体は「機械語」と呼ばれている「コンピュータことば」しか理解できませんが、「MSX BASIC」という、私たちにもコンピュータにも理解できることばを使うことで、コンピュータへの命令が可能になるのです。たとえば、たし算の答を求めるとします。

[P][R][I][N][T]　[1][0][SHIFT]→[+][5][↵]

と入力すれば、MSX BASICがCPUにわかることばになおして伝え、CPUが計算し、その結果を私たちに返してくれるわけです。そこで、MX-101を思いのままにお使いいただくには、このMSX BASICを知らなくてはなりません。
MSX BASICには、文法や約束があり、それを守って命令しますが、基本的なことさえ覚えてしまえば、あとは使っていくうちに自然にマスターできますから、とにかく積極的にMX-101にさわってください。



## 1-3 プログラムとは

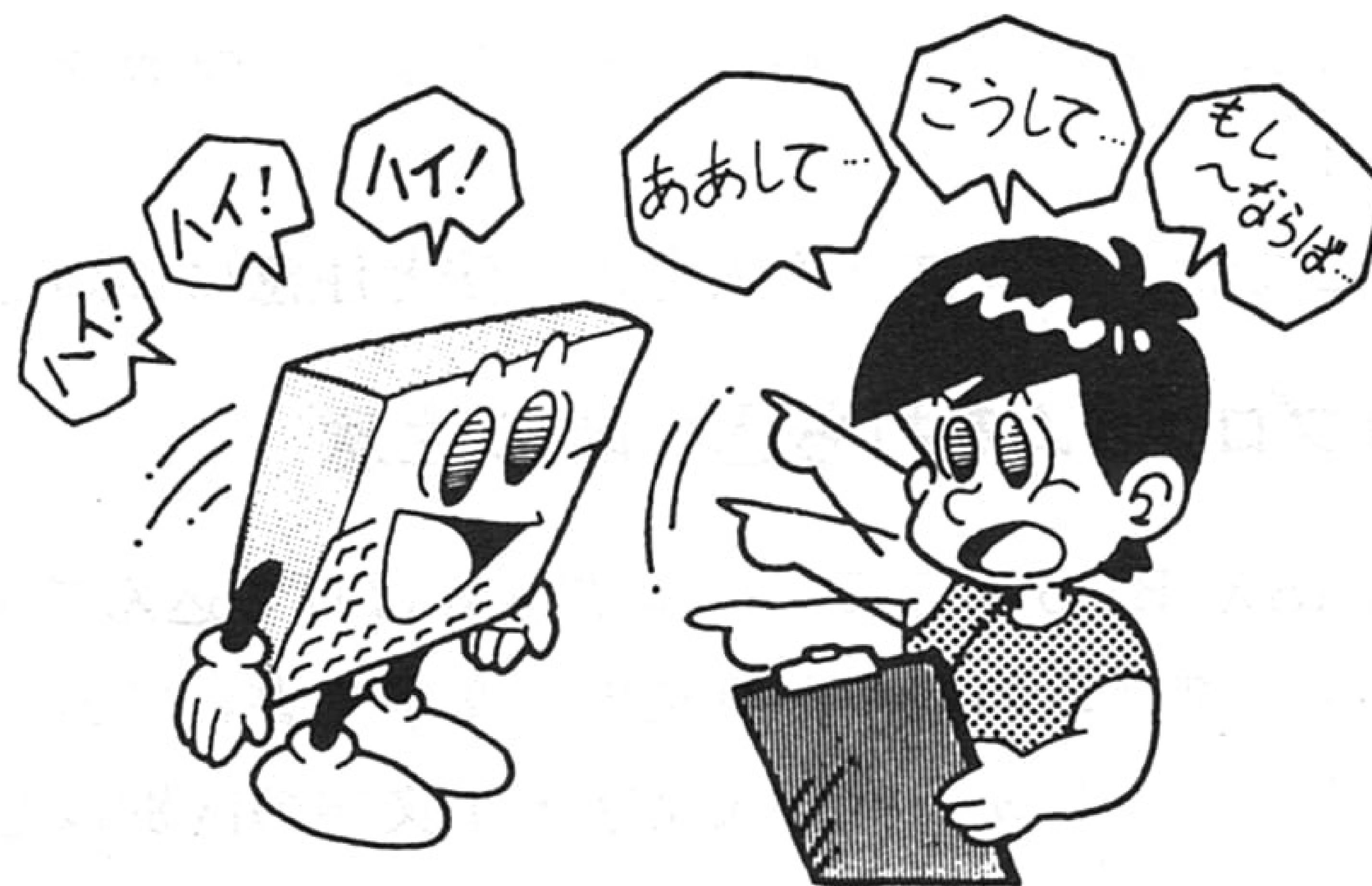
「音楽を演奏しなさい」「計算をしなさい」——これらはすべて命令ですね。人間同士なら、その人にわかることばで言えばいいわけですが、コンピュータに命令するには、プログラムが必要となります。

ところで、命令といってもいろいろあるわけで、たとえば「銀行からお金をおろして、パソコンショップに行き、MSXパソコンを買いなさい。おつりがあったらゲームカートリッジも買ってきなさい」——というのがあったとします。これは、命令の集まりであって、ひとつひとつの命令に分ければ

①「銀行からお金をおろしなさい」
②「パソコンショップに行ってMSXパソコンを買いなさい」
③「もしお金があまっていたならゲームも買いなさい」

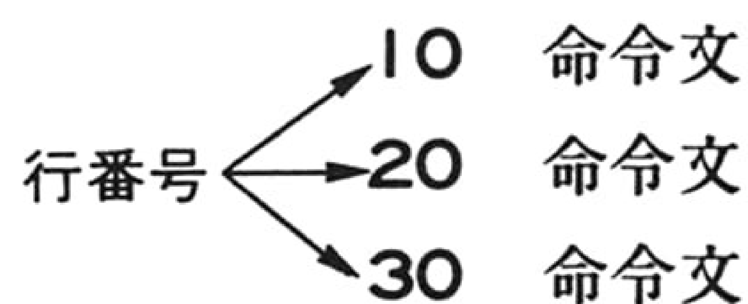
となります。このように「いくつもの命令が集まったもの」がプログラムなのです。

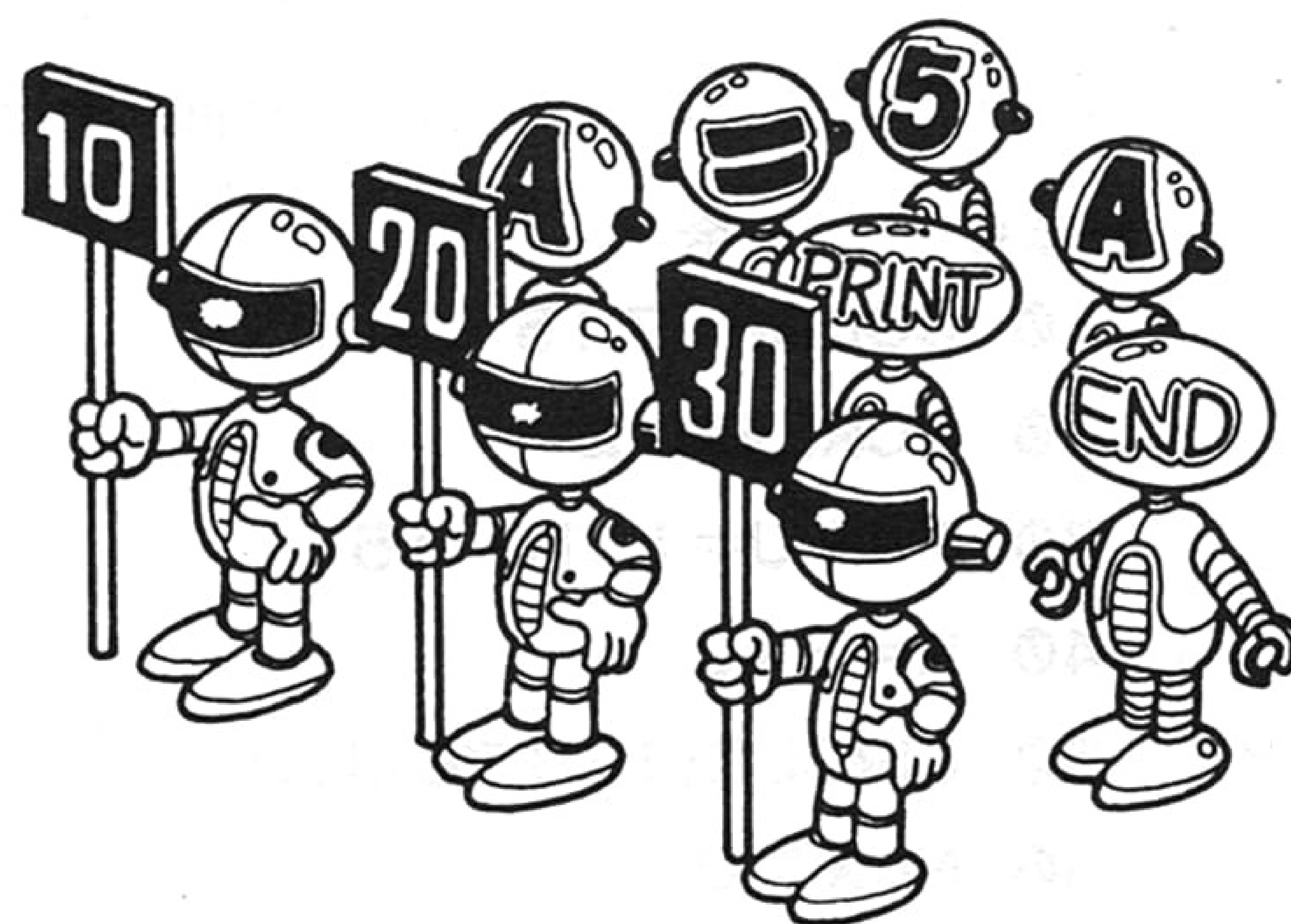
ですから、MX-101用のプログラムとは、私たちとMX-101の共通語「MSX BASIC」の文法や約束にもとづいて書かれた命令の集まりということになります。




この命令には、実行する順番をきめるために、命令のひとつひとつに番号をつけてやる必要があります。これを行番号といいます。

つまりプログラムは、コンピュータに作業の内容とその手順を教えるものなのです。



# 第2章 プログラム入力

## 2-1 プログラムの入力と注意

MSXパソコンにはたくさんのゲームや実用のテープソフトやカートリッジが市販されていますから、それらを使うとすぐにパソコンを使うことができます。それは、テープやカートリッジにすでにプログラムが入っているからです。しかし、プログラムは自分でキーボードから打ち込むことも可能なのです。

ここでは、プログラム入力の方法と注意について説明します。

### ●プログラムを打ち込む前の注意

①MSX BASICでないプログラムは打ち込んでもエラーになります。

②プログラムの文字が英文の大文字で書いてあっても、小文字で打ち込んでもかまいません。なぜなら、コンピュータは大文字と小文字の区別はしないからです。つまり、BOOKもbookもまったく同じとみなしているわけです。でも例外もあります。"(ダブルクォーテーション)でくくられた文字に関しては区別をします。だから〝BOOK〃と〝book〃はちがうのです。

③行番号は、自動的に小さい順にメモリーに整理されますから、打ち込みの順序が逆になってもかまいません。

④まちがって打ち込んでも、打ち込みが終ってから修正できますし、そのまま実行しても画面でまちがいを知らせてくれますから、気軽に打ち込んでみてください。

### ●プログラム入力

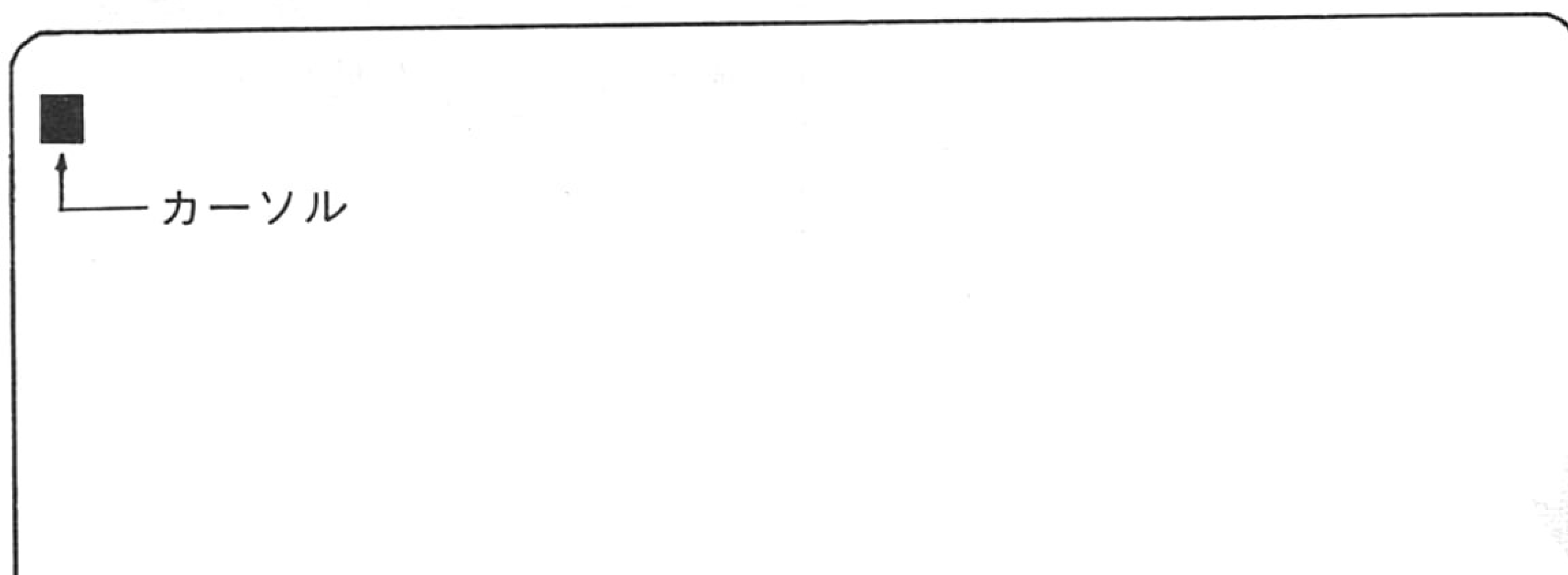
次のプログラムは、たくさんの円を描くためのプログラムです。
まずは、このプログラムを入力して画面に円を描いてみましょう。

```
行番号    命令
 10 COLOR 7,15
 20 SCREEN 2
 30 FOR J=1 TO 15
 40 K=8*J
 50 CIRCLE(128,96),K,,J
 60 NEXT J
 70 END
```

**〈プログラムの入力例〉**

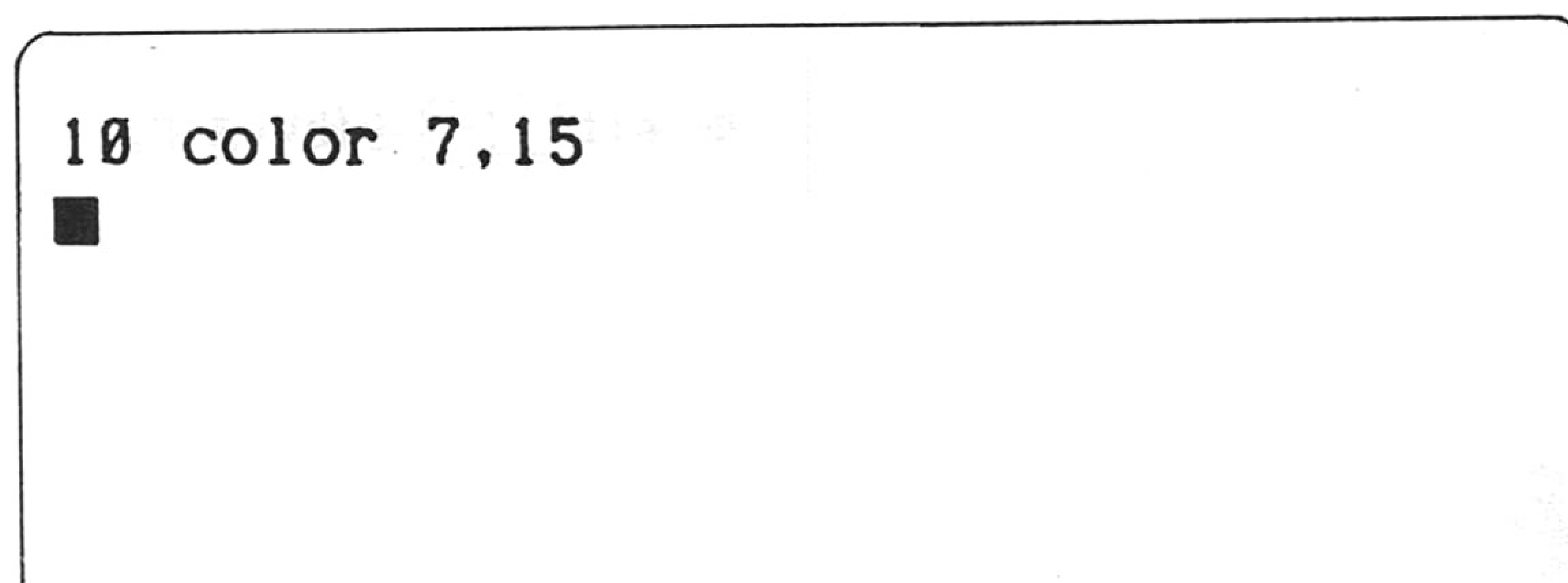
プログラムを入力するとき、英文字は大文字、小文字どちらでもかまいません。
では、なにはともあれ入力してみましょう。

注：コンピュータのなかに大切なプログラムが残っているときはカセットテープへセーブ（44P参照）してから始めてください。
まず、N E W ↵ とキーを押して、コンピュータのメモリーをきれいにします。



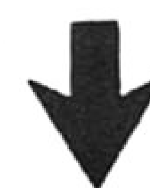

次に画面もきれいにしましょう。SHIFT → CLS HOME で画面をクリアしてから1行ずつ入力していきます。もし入力ミスをしてしまったら、2-2「リスト修正」（38Pから42Pまで）を参照しながら修正してください。

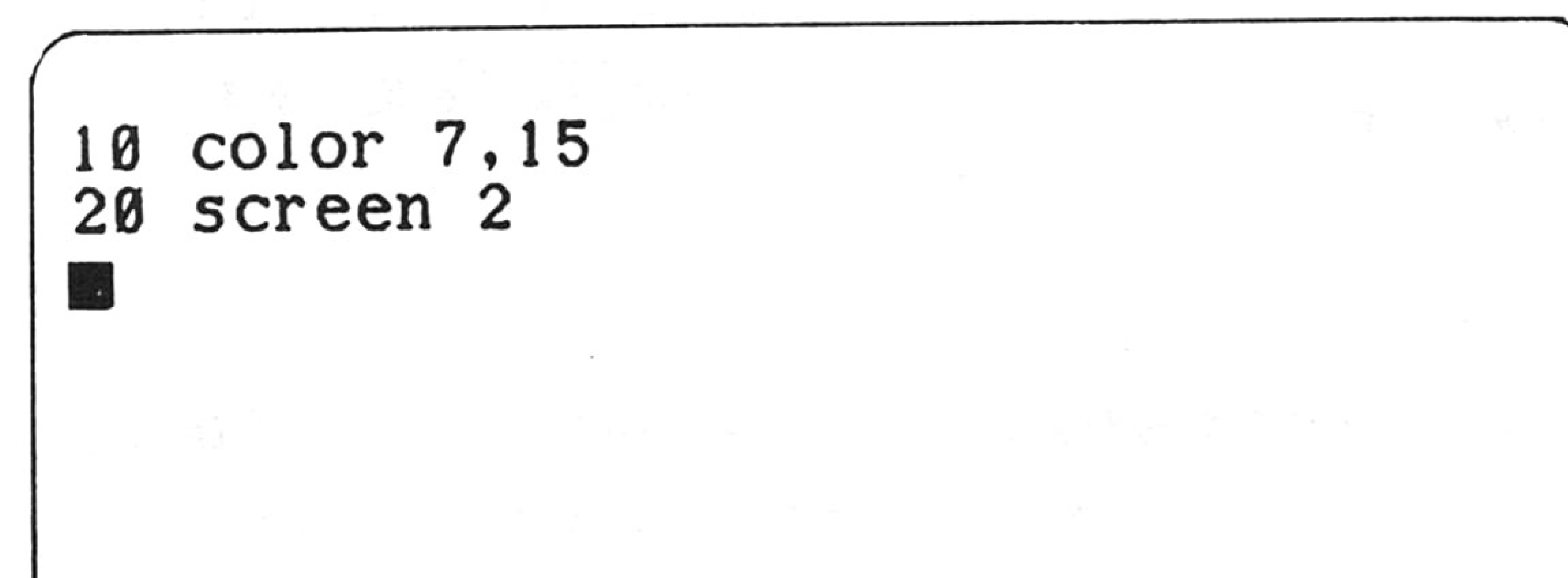





まず10行め（行番号が10の行）を入力します。
1 0 スペース C O L O R スペース 7 , 1 5 ↵

※ 1 0 F1 7 , 1 5 と入力しても同じです。
※ 行の終りに必ず ↵ （リターンキー）を押します。これを忘れるとコンピュータは記憶しません。
※ スペース は押さなくてもかまいません。
※ ，は ，くョ を使います。





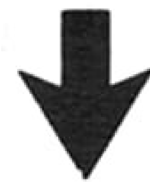

20行めを入力します。
2 0 スペース S C R E E N スペース 2 ↵

```
10 color 7,15
20 screen 2
30 for j=1 to 15
■
```

30行めを入力します。

[3] [0] [スペース] [F] [O] [R] [スペース] [J] [SHIFT] → [=] [1] [スペース] [T] [O] [スペース] [1] [5] [↵]

※＝は [-=] を使います。



```
10 color 7,15
20 screen 2
30 for j=1 to 15
40 k=8*j
■
```

40行めを入力します。

[4] [0] [スペース] [K] [SHIFT] → [=] [8] [SHIFT] → [*] [J] [↵]

※ ＊は [:*] を使います。



```
10 color 7,15
20 screen 2
30 for j=1 to 15
40 k=8*j
50 circle(128,96),k,j
■
```

画面中の＊はアスタリスクマーク(＊)です。

50行めを入力します。

[5] [0] [スペース] [C] [I] [R] [C] [L] [E] [SHIFT] → [(] [1] [2] [8] [,] [9] [6] [SHIFT] → [)] [,] [K] [,] [J]

※(は [8(] を使います。

)は [9)] を使います。

**ご注意**

本書ではプログラムリストや画面表示を見やすくするために、プリンタで印字したリストを使用している箇所もあります。この場合、アスタリスクマーク(＊)が(⁎)となっていますので、入力する際にはご注意ください。

```
10 color 7,15
20 screen 2
30 for j=1 to 15
40 k=8*j
50 circle(128,96),k,j
60 next j
70 end
■
```

60行を入力します。

6 0 スペース N E X T スペース J ↵

最行に70行を入力します。

7 0 スペース E N D ↵

これで、すべてのリスト入力が終りました。どうです、画面は左のようになっていますか？

さて、入力が終ったら、入力されたリストをコンピュータが正しく記憶しているかを確かめます。確かめるにはLIST（リスト）命令を使います。L I S T ↵ または、F4 ↵ と押して、画面のようにリストが表示されれば大成功です。

```
list
10 COLOR 7,15
20 SCREEN 2
30 FOR J=1 TO 15
40 K=8*J
50 CIRCLE(128,96),K,J
60 NEXT J
70 END
Ok
■
```

英文字は、入力のとき小文字であれ大文字であれ、LIST命令のときは大文字で表示されます。

リスト通りに表示されたらF5もしくはR U N ↵でプログラムを実行してみてください。画面にたくさんの円が描かれたでしょう。

次にSHIFT → F6と押します。画面はもとの色に戻ります。（以後、全てのプログラムについて、プログラム実行後もとの色に戻したいときに使ってください。）

**ワンポイント**

**■NEW, LIST, RUN命令**

- NEW命令
  メモリー上にあるプログラムをすべてクリア（削除）します。
- LIST命令
  メモリー上にあるプログラムを表示します。
- RUN命令
  メモリー上にあるプログラムを実行します。RUN命令のあとに、行番号を指定するとプログラムは指定した行から実行します。行番号をつけないときは、プログラムの最初の行から実行します。

**■STOP, END命令**

- STOP命令
  プログラムを途中で停止させるときに使う命令です。
- END命令
  コンピュータに「プログラムの実行を終りなさい」と知らせる命令です。

## 2-2リストの修正

ここまでなんのまちがいもなくスムーズにくればいいのですが、どんなにていねいに打ったつもりでもミスはあるもの。まちがえて打ったり、とばして打ったり……。
でもご安心ください。次の方法で簡単に修正することができます。
このようなプログラムのまちがいを〝バグ〟といい、修正することを〝デバッグ〟といいます。ここでは〝デバッグ〟の方法をケースごとに説明します。ケースとしては

- ●プログラム入力中にミスに気づいた
- ●リスト命令でリストの確認中に気づいた
- ●実行してみたらエラーが表示された

の3つがあります。どの時点で入力ミスに気づいたかによって画面に表示されるリストの状況はちがいますが、訂正、挿入、削除など修正の仕方は同じです。いずれも、修正したいところにカーソルを移動させて修正します。

### ケース1　プログラム入力中にミスに気づいた

訂正）10 color 7，15のcolorをcolarと打ってしまった。

```
10 colar■
```

カーソルをカーソルキー[←]で移動させて、まちがえた文字の上に重ねます。

```
10 colar
```

正しい文字[O]を打ちます

```
10 color
```

カーソルをもとの位置にもどして入力を続けてください。行の入力が終ったら必ず[↵]キーを押します。

削除）20 screen …のscreenをsccreenと打ってしまった。

```
20 sccreen■
```

カーソルをカーソルキー ⬅ で移動し、消したい文字の上にもっていきます。

```
20 sccreen
```

DEL キーを1回押します。そうするとカーソルのある位置の文字が消え同時にカーソルの右側の文字がすべて1文字分ずつ左へ移動します。

※ DEL キーを押し続けると、どんどん文字が消えてしまいますので注意してください。

```
20 screen
```

カーソルをもとの位置にカーソルキー ➡ で移動させて入力を続け行の終りに ↵ キーを忘れずに押してください。

挿入）20 screen……のscreenをscrenと打ってしまった。

```
20 scren■
```

カーソルをカーソルキー ⬅ で移動して文字を入れたいところのすぐ右にもってきます。

```
20 scren
```

INS キーを1回押します。

```
20 scren
```

入れたい文字 E を打ちます。

※ INS キーを押すとカーソルの形が変って "▂" インサートモード（挿入可能な状態）になっています。

```
20 screen
```

➡ キーでカーソルをもとの位置にもどすとカーソルは前の形 "■" にもどります（インサートモード解除）。再び入力を続け、行の終りに ↵ キーを押すことを忘れないでください。

## ケース2　リストの確認中に気づいた

リストの入力を終って、LIST命令で画面にリストを表示させたところ、まちがいを見つけたというときも、やはりカーソルを修正部へ移動させて、ケース1と同じ方法で、訂正、削除、挿入、を行ないます。修正が終ったら[↵]キーを押して修正した行の入力が完了しますが、このときカーソルは修正した行の次の行の先頭に移動します。このまま実行すると"Syntax error"が表示されますので、修正して[↵]キーを押したあと、[SHIFT]→[CLS HOME]でいったん画面をきれいにしてから、再びLIST命令でリストを確かめてから、[F5]または[R][U][N][↵]で実行してください。

部分的な修正はケース1とまったく同じですが、もし1行そっくり抜けていたときは、次のようにします。

例）40行が抜けていたとき

```
10 color 7,15
20 screen 2
30 for j=1 to 15
50 circle (128,96),k,j
60 next j
70 end
```

そのまま40行を打ち込みます。
[4][0][スペース][K][SHIFT]→[=]
[8][SHIFT]→[*][J]

```
10 color 7,15
20 screen 2
30 for j=1 to 15
50 circle (128,96),k,j
60 next j
70 end
40 k=8*j■
```

[↵]キーを押します。
忘れずに！

```
10 color 7,15
20 screen 2
30 for j=1 to 15
50 circle (128,96),k,j
60 next j
70 end
40 k=8*j
■
```

[SHIFT]→[CLS HOME]で画面をクリアーし[L][I][S][T][↵]または[F4][↵]と打って画面にリストを出します。

```
list
10 COLOR 7,15
20 SCREEN 2
30 FOR J=1 TO 15
40 K=8*J
50 CIRCLE (128,96),K,J
60 NEXT J
70 END
Ok
■
```

小文字だったリストは大文字で表われます。入力したとき70行のあとに入れた40行がちゃんと30行と50行の間に入っています。これで大成功です。

なお、1行そのまま削除するときは[行番号][↵]でその行を削除します。

## ケース3 実行したらエラーが表示された

リスト修正が終って F5 (または R U N ↵ ) としたところ、円を描かずに次のような文が出てきたら、まだ入力の誤りがあって実行できないことを意味しています。

例)

```
run
Syntax error in 40
```

これは「40行目にsyntax error (入力ミス)がありますよ」というコンピュータからの知らせです。このように、修正する行番号とエラーの内容が表示されますから、その行番号を画面に呼び出して修正すればいいのです。リストを画面に呼び出すにはLIST命令を使いますが、このLIST命令は使い方により、特定の行を呼び出したり、呼び出す範囲を指定したりできます。Syntax error の場合は、エラー行を自動的に呼び出す L I S T . ↵ という、とても便利な方法があります。

例) プログラム入力を2ヵ所まちがえたまま F5 で実行した。

```
10 COLOR 7,15
20 SCREEN 2
30 FOR J=1 TO 15
40 K=8:J          ―:が間違っている
50 CIRCLE (128,96),K,J
60 NET J          ―Xが抜けている。
70 END
```

F5 で実行します

```
Syntax error in 40
Ok
■
```

40行に入力ミスがあるので L I S T . ↵ と入力してエラー行を表示させます。
※・は ・ を使います。

```
list
40 K=8▒J          ―*に訂正
Ok
```

カーソルを:に移動して SHIFT → * と押します。訂正されたら ↵ を押し、再び F5 で実行します。

```
Syntax error in 60
Ok
■
```

またエラーが出ました。今度は60行です。そこでまた L I S T . ↵ でエラー行を表示させます。

```
list.
60 NET          ―Xを追加する
Ok
■
```

カーソルをTに移し、 INS キーを押してインサートモードにし、Xを挿入します。

挿入が終ったら、忘れずに ↵ を押し、 F5 で実行します。

これで、もうエラーはないので、画面に円が描けるはずです。
プログラムが終了したら、SHIFT →F6 と押して、画面をもとの色にもどします。
このようにデバッグにはLIST命令が欠かせません。次の表を参考にして、LIST命令を使い分けてください。

## LIST命令の使い方

| 打ち方 | 意味 |
|---|---|
| L I S T ↵ または F4 ↵ | プログラム全部を表示します |
| L I S T . ↵ または F4 . ↵ | エラーのあった行を表示します |
| L I S T ×× ↵ または F4 ×× ↵ | ××行を表示します |
| L I S T ××－ ↵ または F4 ××－ ↵ | ××行から終りまで表示します |
| L I S T ××－△△ ↵ または F4 ××－△△ ↵ | ××行から△△行まで表示します |
| L I S T －×× ↵ または F4 －×× ↵ | 初めから××行まで表示します |

(××や△△には行番号が入ります)

行番号だけを入力し、↵ キーを押せば、その行の内容全てが消えます。また、同じ行番号を持つ別の内容を ↵ キーで記憶させると、後から記憶させた方に変わります。

(例) 10 COLOR 7,15 ↵
10 PRINT 123 ↵
と操作後、LIST ↵ と操作してみましょう。

# 第3章　カセットテープによるプログラムの保存

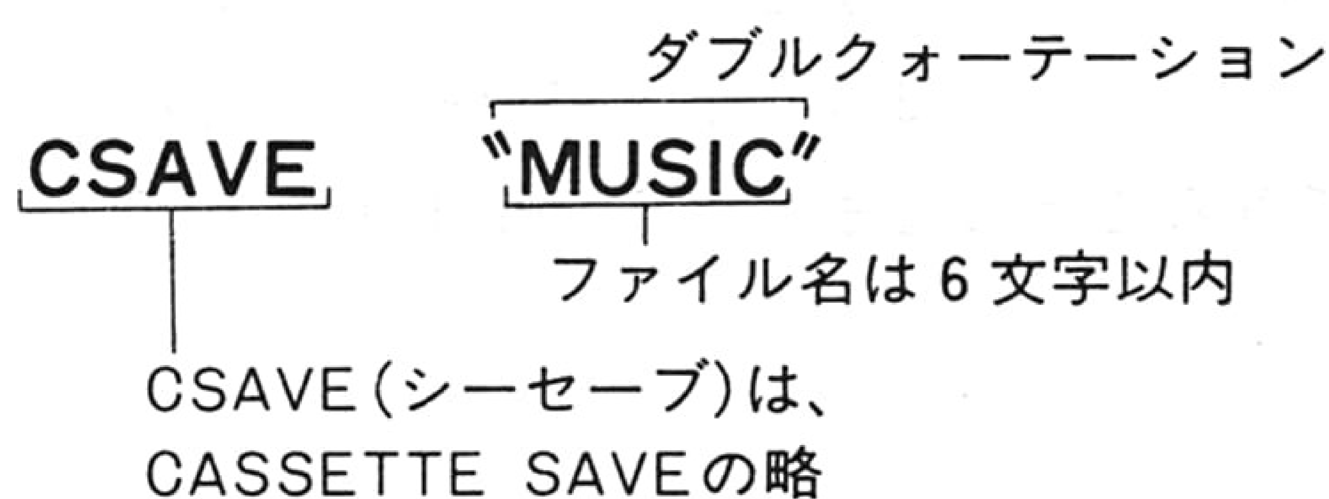
せっかく苦労してプログラムを入力しても、コンピュータの電源を切るとプログラムはすべて消えてしまいます。
そこで、ここでは入力したプログラムをカセットテープに録音したり、また逆にカセットテープに録音されたプログラムを読み込む､「セーブ」と「ロード」について説明します。
カセットテープを利用するというと、ゲームなどの〝テープソフト〟を思い出すでしょう。これから行なうカセットテープへの録音は、つまりこの「テープソフト」を自分で作ることなのです。そして、プログラムを録音することをSAVE(セーブ)するといい、カセットテープのプログラムを再びコンピュータに読み込ませることをLOAD(ロード)するといいます。ここでは練習のため、ドレミファソラシの7音階の音を出すプログラムを録音してみることにしましょう。まず次のように入力してください。

```
10  play  "cdefgab"
20  end
```

さて、録音するプログラムには名前（ファイル名ともいいます）をつける必要があります。これはロードするときに、コンピュータがそのプログラムを見つけるための目印と考えてください。
そして、名前は6文字までで　"　（ダブルクォーテーション）で囲むことを忘れないでください。また名前はアルファベットの大・小文字、数字、かななどのどれでも使えますので好きな名前をつけることができますが、プログラムの内容がわかるような名前にしておくと、あとでさがすときに便利です。
ここでは、いま入力したプログラムを「MUSIC」という名前にします。

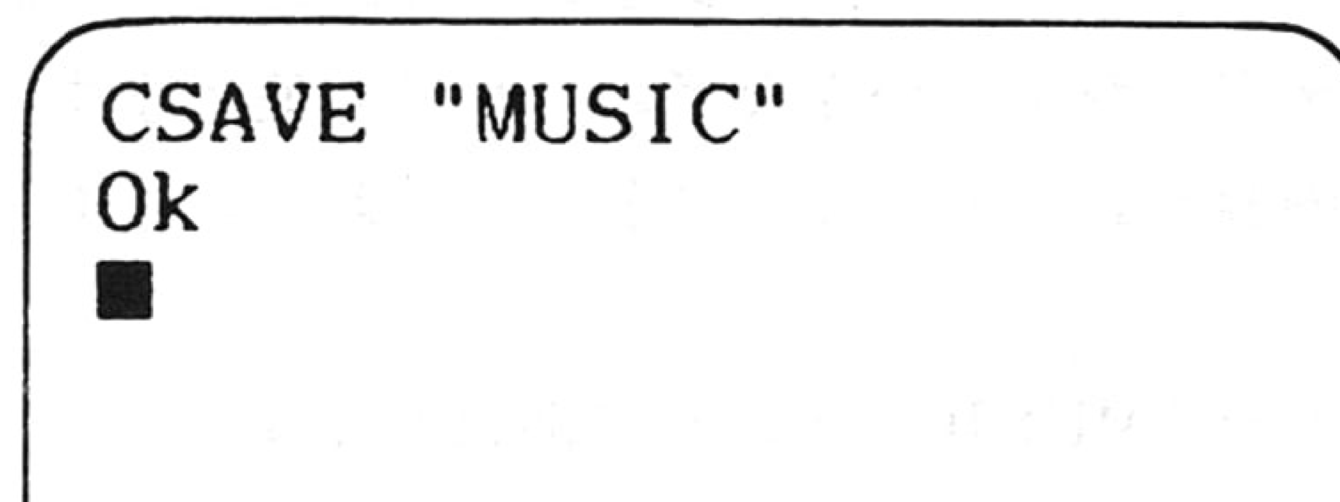
## 3-1 セーブの仕方(プログラム録音)

1. カセットテープレコーダーをセットします。(11Pのカセットテープレコーダーの接続を参照)
2. カセットテープを用意します。15～30分の短いものでいいのですが、なるべく新しいものにしましょう。
3. カセットテープをテープレコーダーにセットし、リセットボタンを押してテープカウンターを〝000〟にもどしておきます。
4. リーダーテープ(はじめの白い部分)を早送りして巻き取ります。
5. 音量(ボリューム)を最大、またはその近くに合わせます。
6. テープレコーダーの録音ボタンを押します。(テープはまだ回りません)
7. コンピュータに記録の命令をします。それは次のようにします。




これで準備は終りです。

8. ⏎キーを押します。カセットテープが回りはじめ、セーブを始めます。
9. セーブが終るとカセットテープはとまり、次のように表示されます。




※テープレコーダーに、リモート端子がついていないときは、テープレコーダーの回転をスタートさせたり、ストップしたりする手作業が必要です。このときは録音をスタートさせてから⏎キーを押し、OKが表示されたらストップさせてください。

## 3-2 正しくセーブできたかをチェックしましょう

プログラムをセーブしたときは、コンピュータの電源を切る前に、正しくセーブされているかチェックする習慣をつけましょう。

1. テープを巻きもどします。(テープカウンターを目安に少し手前までもどします)
2. 再生ボタンを押します。
3. 次のように入力します。

**CLOAD ? 〝MUSIC〟**

チェックすることをVerify(ベリファイ)といいますが、
これはベリファイと同じ動作をする命令です。

※ファイル名は、セーブしたときの名前の大文字、小文字などに注意し、正しく入力してください。

4. [↵] キーを押します。カセットテープが回りはじめ、MUSICという名前を読みとると、次のように表示します。

```
CLOAD?"MUSIC"
Found:MUSIC
```

この表示のあと、録音したプログラムを読み込み、コンピュータのプログラムと比較してチェックします。

5. セーブが正しく行なわれているときは〝Ok〟と表示されます。
6. チェックしても〝Ok〟がでないとき、例えば、

```
Skip :MUSIC
```

のときはチェックのファイル名をまちがえて入力しています。大文字、小文字も正確に入力しましょう。(ファイル名は大文字、小文字の区別をします)

```
Verify error
```

のときは正しく録音されていません。次のことをチェックしましょう。

- テープレコーダーの接続は正しくされていますか。
- 音量(ボリューム)は最大になっていますか。
- カセットテープが傷ついていませんか。
- テープレコーダーのヘッドが汚れていませんか。

また、ファイル名が同じでも、プログラム内容がちがっているときにも、このメッセージが出ます。エラーの内容を確かめて、再度セーブしてください。

## 3-3 ロードの仕方 (プログラムの読み込み)

カセットテープに録音したプログラムは、自作テープソフトとして何度でも使うことができます。これをコンピュータに読み込ませるときは、「テープソフト」を使うのと、ほぼ同様に行なえます。

1. カセットテープを、録音されている部分の少し手前にセットします。1本のテープにいくつものプログラムを録音するときは、テープカウンターで頭出しのカウントを記録しておくと、このようなときに便利です。
2. 音量(ボリューム)を最大にします。
3. テープレコーダーの再生ボタンを押します。
4. コンピュータに読み込むための命令を入力します。
   さきほど録音した〝MUSIC〟をロードしてみましょう。

   CLOAD 〝MUSIC〟

ファイル名は、セーブしたときとまったく同じ文字でなければなりません。

また、チェック命令であるCLOAD？〝ファイル名〟と、CLOAD〝ファイル名〟とを区別してください。

5. [↵]キーを押します。カセットテープは回りはじめます。
6. 〝MUSIC〟というプログラムが見つかると〝Found：MUSIC〟と表示されます。
7. ロードが終ると〝Ok〟と表示されます。
8. ロードが終了したら、さあ実行です。

   [F5]（または [R][U][N][↵]）

   どうですか、ドレミファソラシと音が出たでしょう。

（注：テレビの音量がしぼってあると、聞えません。）

※ロードするときファイル名を省略すると、最初に出会ったプログラムを読み込みます。また、カセットテープの頭出しをしないときは、目的のプログラムをみつけるとロードし、目的のプログラムがないときは、最後まで進んでしまいます。

※リモート端子がない場合は[↵]キーを押してから再生をスタートさせます。

**ワンポイント**

**■セーブをするときには……**

テープのどこから録音したか、カウンタの目もりとそのプログラムにつけたファイル名をメモして、テープといっしょに保管しておくことをおすすめします。こうすれば、目的のプログラムはすぐに見つかりますね。

# 第4章　プログラミング講座

## 4-1 フローチャート（流れ図）

コンピュータはプログラム（行番号をつけた命令の集まり）によって、いろいろな作業をしますが、その作業の手順をわかりやすく図式化したものを「フローチャート」といいます。簡単なプログラムなら、わざわざフローチャートを書く必要はありませんが、慣れないうちはフローチャートに作業全体の手順を整理しておくとプログラミングしやすくなります。
フローチャートを書くことによって、行ごとの処理の内容や順序もはっきりし、手落ちやまちがった処理をしなくてすむわけです。また、あとでプログラムを手直しするときにも大いに役立ちますし、友だちとプログラムを交換するときにも便利です。

### (1) フローチャート記号

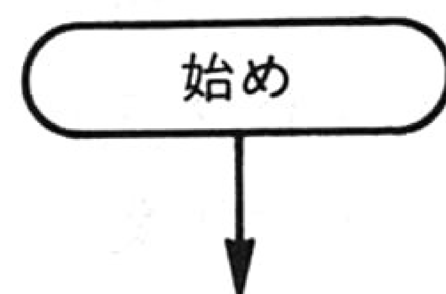
フローチャートを書くためには、いろいろな記号があります。特にこの記号を使わなければならないということではないのですが、一応の基準がありますので、ここでその記号の説明をしておきます。
フローチャートは、これらの記号を線で結んで書きあらわします。

- 端　子　プログラムの〝始め〟と〝終り〟をあらわします。
- 準　備　プログラムを作るときに、変数の初期値設定に使います。
- 手入力　数字や文字をキーボードから入力するときに使います。
- 表　示　画面に表示するとき、画面設定などに使います。
- 書　類　プリンタやプロッタに打ちだすときに使います。
- 判　断　条件に合っているかどうか判断します。
- 処　理　計算などの処理に使います。
- サブルーチン　サブルーチン呼び出しに使います。
- 結合子　フローチャートが長すぎる場合、途中で切るときに使います。また、線が混みいったときにも使います。この丸の中にA、B、C…や1、2、3…という具合に、英字や数字を入れ、つながっていることを表わします。
- 矢　印　プログラムの流れる方向を示します。フローチャートは通常上から下へ流れますが、方向を変るとき（下から上へ行くときなど）は必ず矢印を書きます。

## (2) フローチャートを作ってみましょう

**〔たし算のフローチャート〕**

では単純なフローチャートを作ってみましょう。
ここではキーボードから入力された、2つのデータのたし算を行ない、その結果を表示します。



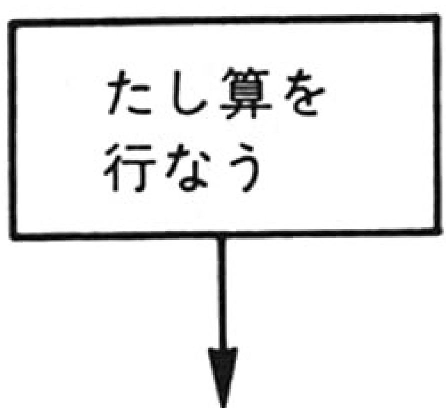

①プログラムの開始を示す記号を書きます。そこから下に線を引き、流れがあることを示します。




②次はデータの入力を表わす記号をつけ加えます。



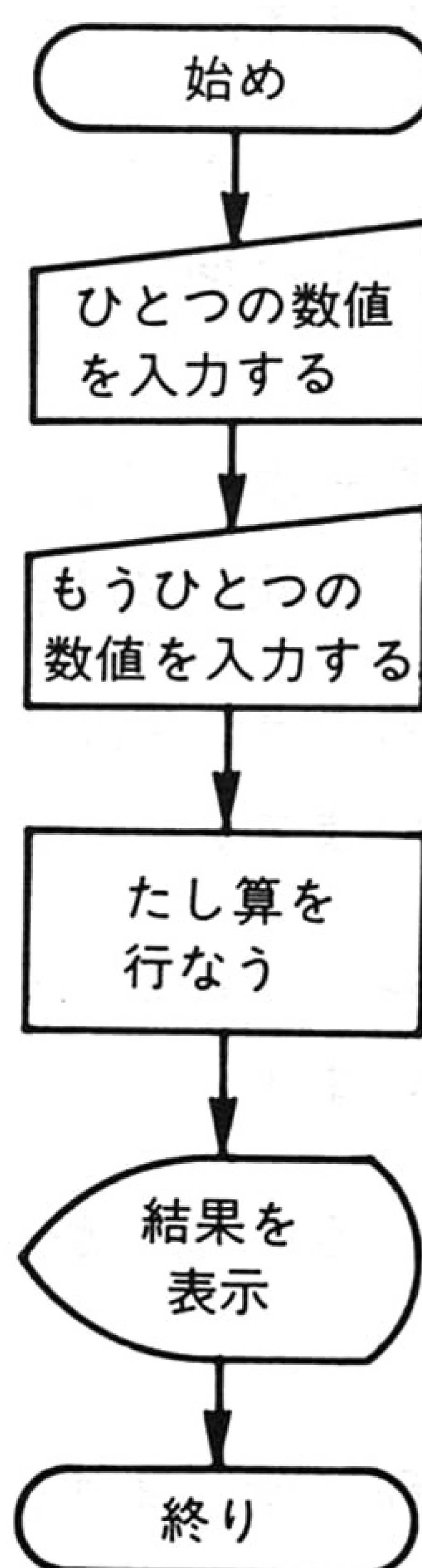

③さらにもうひとつの数字入力を表わす記号を追加します。




④そのあとに、計算を行なう記号を書きます。



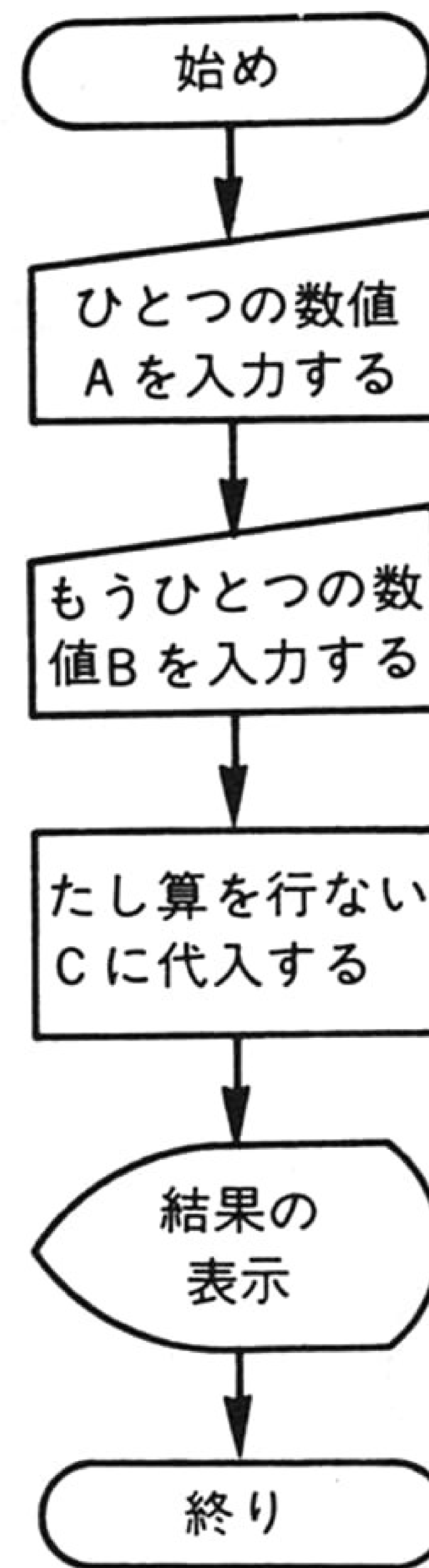

⑤さらに、結果を表示する記号を書き加えます。




⑥これで処理が終了するので、終りを表わす記号を書き、ひとつのフローチャートができあがります。

**●プログラム例**

```
10 INPUT "A=";A
20 INPUT "B=";B
30 C=A+B
40 PRINT "A+B=";C
50 END
```




10 INPUT 〝A=〃;A

20 INPUT 〝B=〃;B

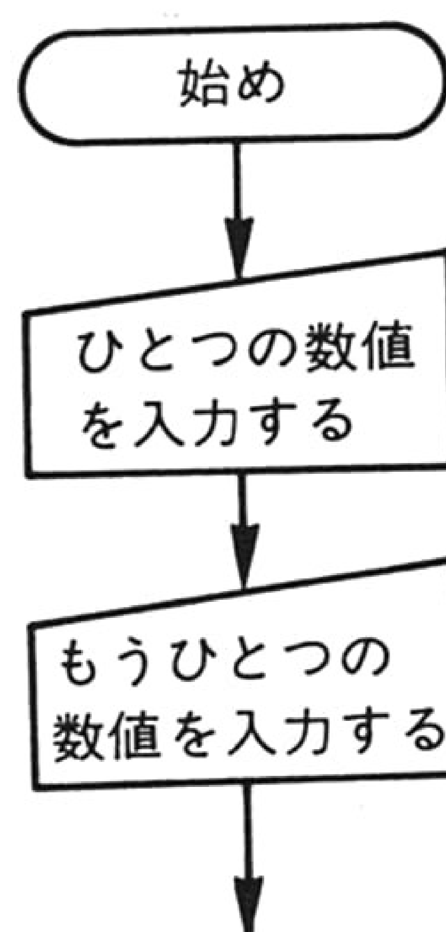
30 C=A+B

40 PRINT 〝A+B=〃;C

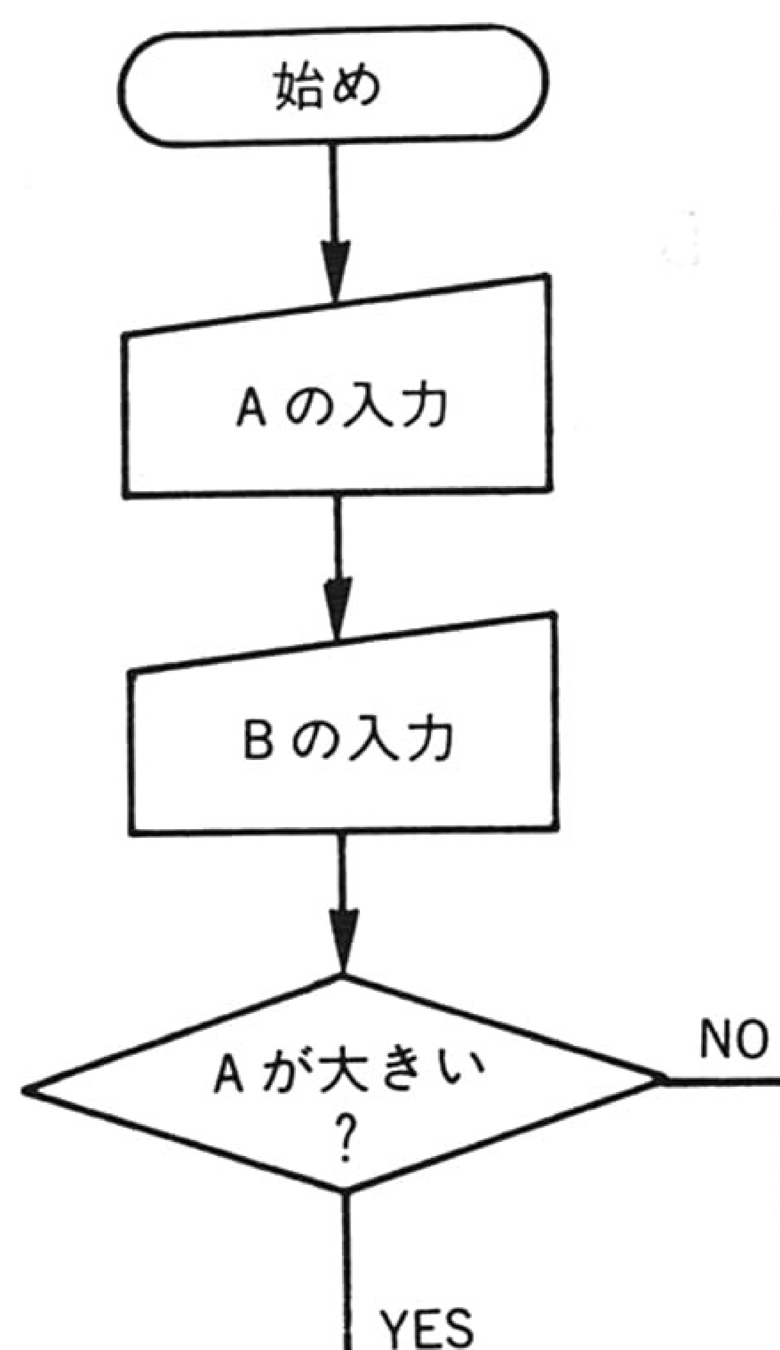
50 END

**〔比較のフローチャート〕**

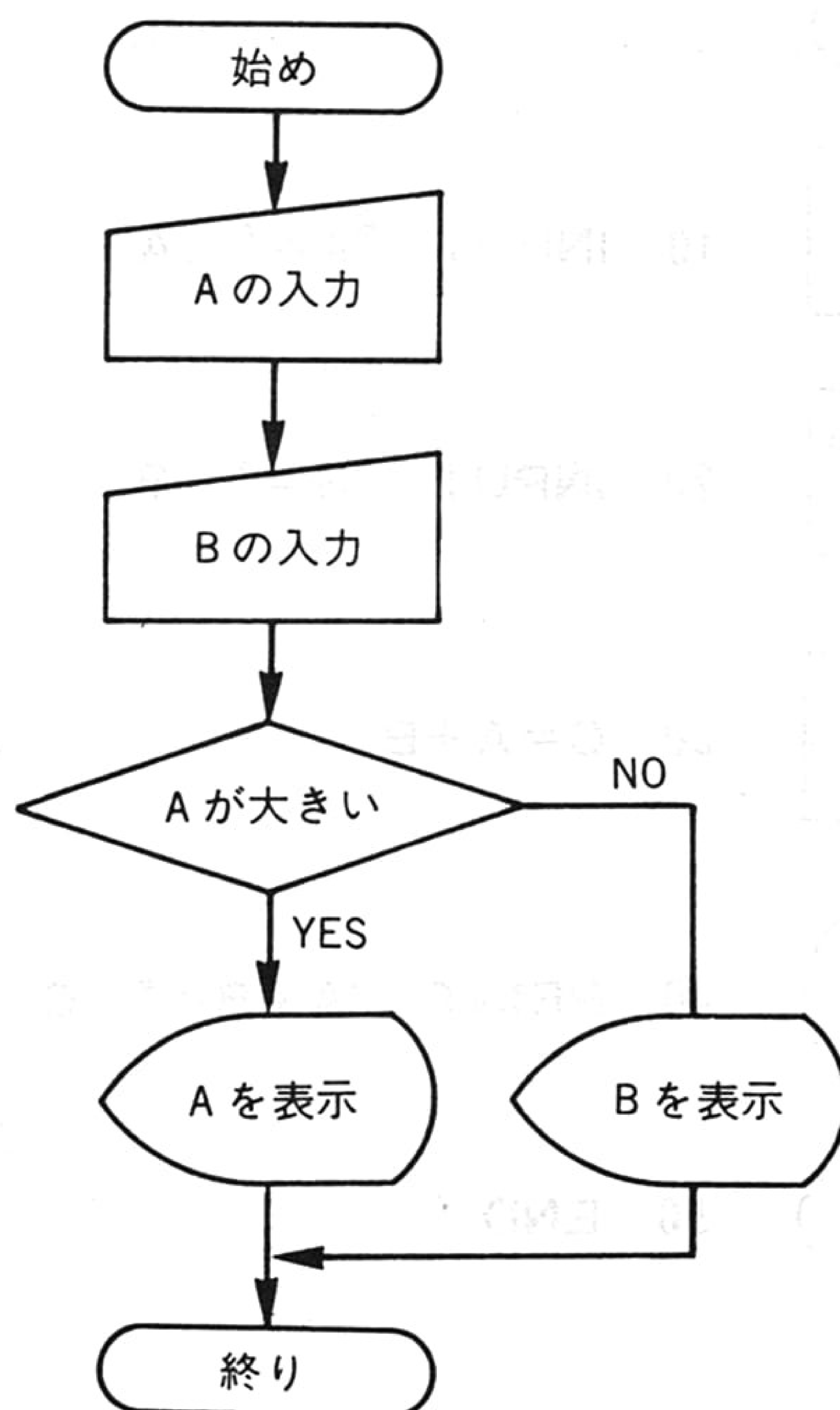
次に入力した2つの値を比較して、大きい値を表示するプログラムを考えてみましょう。




①このように、2つの数値を入力するところまではまったく同じです。
最初の数値を記憶する変数をAとし、次の数値をBとします。




②次の2つの数値を比べて大きい方を表示する部分を考えてみましょう。
比較の記号を書いたあと、流れを表わす線を2本書きます。それぞれ、YES、NOによってわかれます。比較記号のなかは〝Aが大きい?〃という文にします。




③表示部分と終り部分を追加し、フローチャートを完成させます。
どちらの表示も終りはひとつです。

プログラム例……フローチャートとよく見比べてください。

```
10 INPUT "A=";A
20 INPUT "B=";B
30 IF A>B THEN 40 ELSE 60
40 PRINT A
50 GOTO 70
60 PRINT B
70 END
```

下のプログラムも、同じフローチャートをもとに作ったものです。

```
10 INPUT "A=";A
20 INPUT "B=";B
30 IF A>B THEN PRINT A ELSE PRINT B
40 END
```

## 4-2 虫を殺そう（デバッグの方法）

プログラムが正しく走らなかったり、結果がおかしかったときは、プログラムのどこかに必ずミスがあります。このミスのことを「バグ(虫)」といいます。このバグを見つけ、正しく動作するように修正することを「デバッグ(虫を殺す)」といいます。このデバッグの方法をしっかり覚えて要領よくやれば、プログラム完成までの時間をぐんと短縮することができます。

### (1) プログラムが走らないときは入力を疑え！




プログラムが走らないときの原因の第一は、なんといっても入力ミスです。自分では正しく入力したつもりでも、意外とミスがあるものなのです。
入力ミスのときは、〝Syntax error〞が表示され、エラーした行番号まで出ますから、その行をよく見てください。案外、簡単なつづりミスかもしれません。どうしてもまちがいがわからないときは、〝リファレンスブック〞を見て、各命令の正しい規則を確認してください。

### (2) エラーが発生した行以外に誤りがある場合

Syntax error以外のエラーが発生した場合は、その行に必ずしもまちがいがあるとは限りません。〝Illegal function call〞エラーなどは、特に注意が必要です。
たとえば、右のプログラムなどは、30行で、〝Illegal function call〞エラーになりますが、まちがえているのは、実は10行なのです。

```
10 X=-10
20 Y=25
30 Z=SQR(X)+SQR(Y)
40 PRINT Z
```

平方根を求めるSQR関数は負の数を引数とすることができないため、SQR(X)つまり$\sqrt{-10}$が計算できないからです。
このようなときは、30行で使われている変数の内容をPRINT文などで直接確かめてみることが大切です。

```
PRINT X, Y ↵
```

とすれば、それぞれの内容が表示されます。長いプログラムのときは、エラーがおきた行の変数の内容を調べ、その変数が使われている行を次々に確かめていきます。

### (3) エラーは表示されないが、正しい結果にならなかったとき

このようなときは、プログラムの流れに問題があります。プログラムの流れを確かめたいときは〝TRON〟を使いトレースモードにします。TRON↵としたあとで、RUNを実行すれば、実行した行番号が次から次へと表示されます。表示を止めたいときはSTOPキーを押します。TROFF↵でトレースモードは解除されます。TRONはテキストモードでしか意味をもちません。また画面表示が乱れますので、注意して使ってください。(138P参照)

**ワンポイント**

専門用語？　俗語？　……パソコンを使っていると不思議な言葉がたくさんでてきます。その言葉の意味がわかれば、友だちとの話が楽しくなります。

「走る」………プログラムを実行すること
「通る」………プログラムの一部が正しく実行されること
「飛ぶ」………プログラムの実行がGOTO文などによって、行番号順に実行しないで離れた行番号に進むこと。

ところでエラーを発見したら、とことん自分で考えてみましょう。エラーをひとつ直せたら一歩前進、プログラム学習のコツです。どこが悪いのかわかるようになれば、立派に一人前です。

# 4-3 変数

## (1) 変数の意味

プログラムを作る大事な要素として変数があります。変数というのは、ひとくちにいうと数値や文字列を貯わえておく箱といえます。この変数の使い方を理解しないと、せっかくのMX-101もただの電卓になってしまいます。

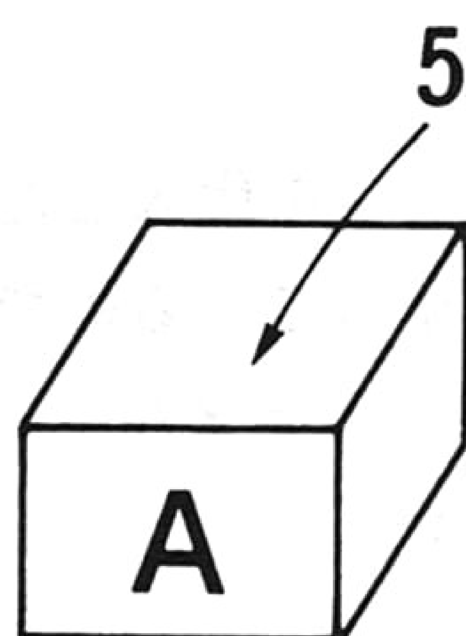
P R I N T 5 SHIFT + 1 0 ↵

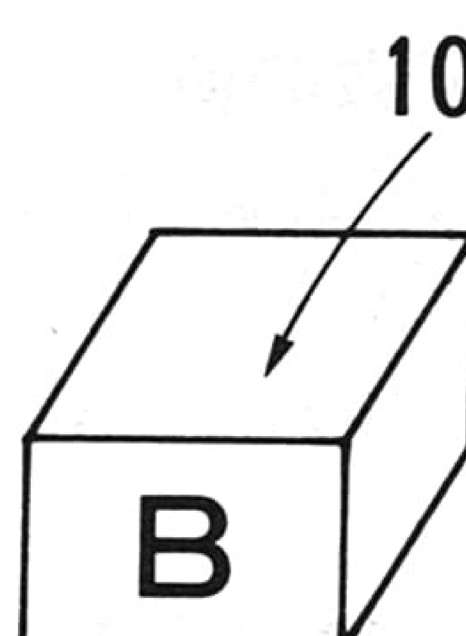
とすれば、画面上に15が表示されます。

では次のように入力してみてください。

A SHIFT = 5 ↵
B SHIFT = 1 0 ↵
P R I N T スペース A SHIFT + B ↵

これも画面に15が表示されます。A＝、B＝、の「＝」は、イコールという意味ではなく、「左辺に右辺を代入する」という意味です。Aのなかには5、Bのなかには10が入っていましたので結果が15になったというわけです。
このA、Bという箱を変数といいます。変数には数字を記憶する数値変数と、文字列を記憶する文字変数があります。



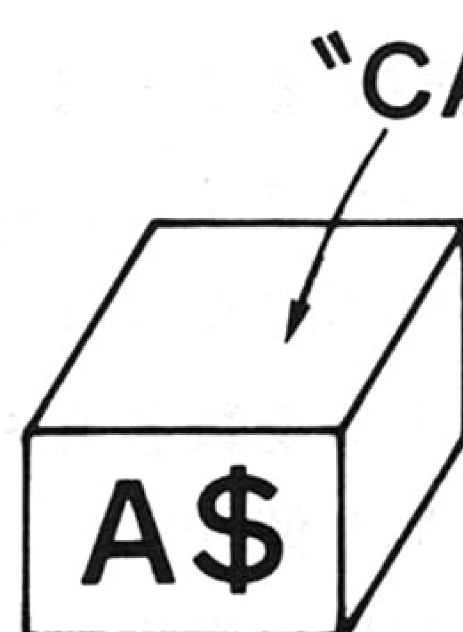



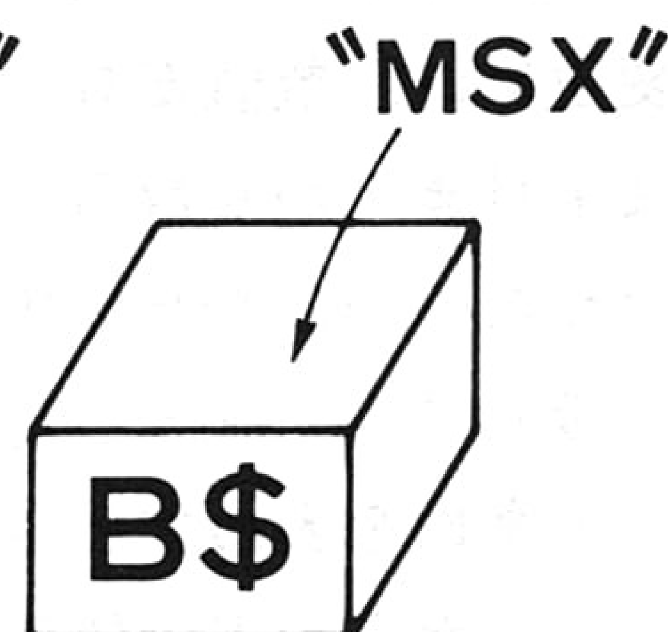

ふつうの計算式は、5＋10＝と書きますが、BASICでは、変数名＝5＋10と書くことができます。これは、5と10をたした結果が変数のなかに入るということです。

| | |
|---|---|
| 5＋10＝A | 正しくない |
| A＝5＋10 | 正しい |
| A＝A＋5 | 正しい（Aの内容に5をたして、その結果を再びAに入れる） |

変数は、前に説明したように、文字も記憶します。

A$＝"CASIO" ↵
（"CASIO"——文字列といいます。）
B$＝"MSX" ↵
PRINT A$＋B$ ↵
「CASIOMSX」と表示されます




文字列はダブルクォーテーション（"）で囲みます。このとき変数名の最後は$記号をつけ、文字を取扱う変数にします。
※ダブルクォーテーション（"）は SHIFT を押しながら 2 を押します。

**まとめ**

数値を代入する（記憶する）………数値変数
文字を代入する……………………文字変数
（例）A$＝"CASIO"

## (2) 変数名のつけ方

変数には自由に名前をつけることができます。この名前を「変数名」と呼びます。たとえば、「A＝10」と書いたときはAが変数名になります。
名前をつけるのは自由ですが、一応変数名のつけ方には次のような規則があります。

①変数名の先頭1文字はアルファベット（A～Z）の文字で始まり、1文字か2文字の長さになります。2文字目はアルファベット以外に数字（1～9）も使うことができます。3文字以上の変数名をつけてもエラーにはなりませんが、区別はされません。

KO＝10
KOTAE＝10
KOKUGO＝10
} すべてKOという変数名です。

②変数名のなかに予約語(BASICの命令や関数名)を含むことはできません。(251P参照)

AP＝10……正しい
APRINT＝10……〝PRINT〞という予約語が含まれています。

## (3) 変数の値

変数は、数値を代入したり、文字を代入したりすると、そのときの値を記憶しますが、何も記憶させないときは、どのような値になっているのでしょうか？

文字変数のとき…文字変数のときヌルコード（〝 〞）といって、何も入っていません。文字列の長さは0です。

数値変数のとき…0 です。

```
new          ←――― まず、new [↵] でメモリーをクリアーしてから次のリストを入力。
Ok
10 A$="CASIO" }←― CASIOのOとMSXとの間に何もない
20 B$="MSX"   }   つまりD$は何も値を代入していない状態です。
30 C$=A$+D$+B$
40 PRINT C$
50 END
run          ←――― [F5] またはrun [↵] で実行します。
CASIOMSX     ←――― "CASIO"と"MSX"が続けて表示されます。
Ok
```

## (4) よく使われる変数名

変数には、数値変数と文字変数があることはもうおわかりだと思いますが、変数は英字で始まり、2文字目までが変数名として区別されます。この変数の名前は意味なくつけるより、ある程度意味をもたせた方がわかりやすいプログラムになります。
一般的によく決まった使い方をする変数名もあります。

①座標を表わすときは、XとYを使う

画面上の表示位置を表わす座標の指定はX(ヨコ)とY(タテ)を使います。座標の位置が2つ以上あるときはX1、X2……、Y1、Y2……のようにします。

②FOR文の制御変数はJを使う

FOR文の制御変数(カウント変数)は、カウントが終ると使わなくなるときがあります。他のFOR文を実行するとき、その変数を使えば余分なメモリーを使わなくてすむので便利です。このようなことから、FOR文で制御変数だけとして使う変数はJを使うようにしましょう。
また、ループが二重になるときは、K、L を使うとよいでしょう。

**ワンポイント**

変数名は属性記号を使うことにより、取扱うことのできる値が変わります。

| | | |
|---|---|---|
| 変数名% | 整数型 | …………－32768～32767までの整数<br>(範囲外の数値は〝Overflow〟エラーとなり代入できません) |
| 変数名! | 単精度実数 | …… 6桁以下の実数 (オーバーした数値はまるめられ指数表示となります。) |
| 変数名# | 倍精度実数 | ……14桁以下の実数 (オーバーした数値はまるめられ指数表示となります。) |
| 変数名$ | 文字列 | …………255文字以内の文字<br>(256文字以上は〝String too long〟エラーとなります) |

属性記号をつけないと倍精度実数になります。取扱う数値が整数だけのときは、なるべく整数型にすればメモリーの消費が倍精度実数よりも少なくてすみます。さらに処理速度も速くなります。

プログラムの最初にDEFINT A-Zを書いておけば、属性記号（%）をつけなくても、変数は全て整数型になります。

# 4-4 配列変数

## (1) 配列の考え方

ひとつの変数では、いままでひとつの値しか記憶できませんでした。しかし、配列変数を利用することによって、ひとつの変数で多くの値を記憶することができます。たとえば30人の名前を記憶するときには、どのようにすればよいでしょうか？

①配列を使うとき

A＄(1)＝〝アライ〟
A＄(2)＝〝イモト〟
A＄(3)＝〝オノ〟
⋮
A＄(30)＝〝ワダ〟

配列変数を使えば変数名がA＄ひとつですむ。

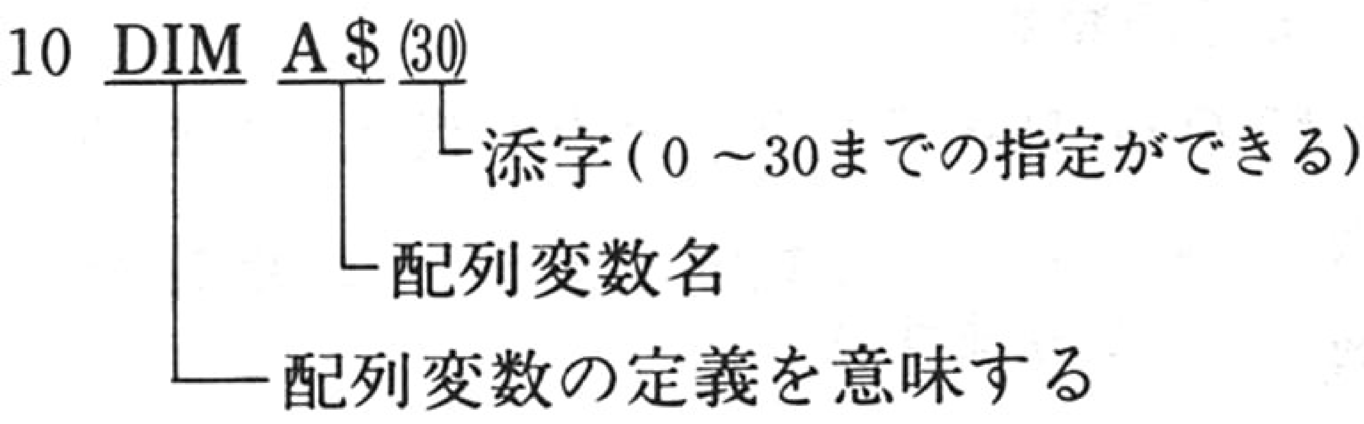
②配列を使わないとき

A＄＝〝アライ〟
B＄＝〝イモト〟
C＄＝〝オノ〟

配列変数を使わないときは30人分の変数を考えて、使わなければいけない……非常に大変です。

## (2) 配列変数の指定

配列変数を使うときは必ずDIM文で配列を使う宣言をします。




"A$(n)"のカッコのなかの数字nは配列変数の大きさを意味します。30とすれば、A$(0)～A$(30)の変数が使えることを意味します。

数値を使いたいときは、配列変数名を数値変数名にします。

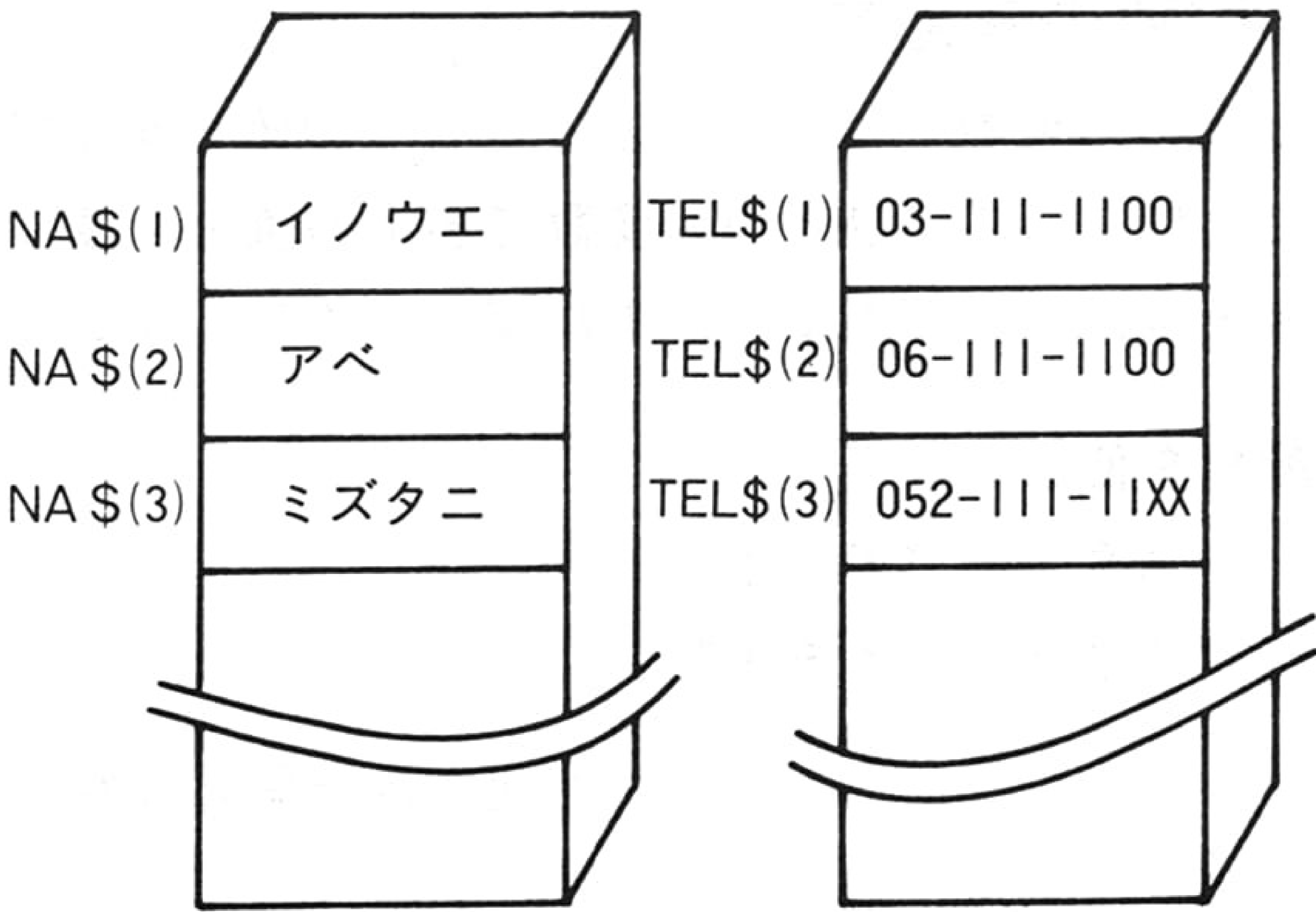
```
10 DIM A(50)
```

とすれば51(0～50)個の数値を記憶できます。この配列変数名のつけ方は、普通の変数のつけ方と同じです。

## (3) 配列変数を使ってみましょう

配列変数を使って電話メモを作ってみることにしましょう。まず、電話メモを図にして考えてみます。

電話メモには、人の名前と電話番号が必要です。その名前と電話番号を記憶する配列変数を箱として考えて図にしてみました。名前を記憶する配列名をNA$とし、電話番号を記憶する配列名をTEL$とします。もちろんTEL$はTE$と同じです（2文字だけ区別するため)。



## プログラム

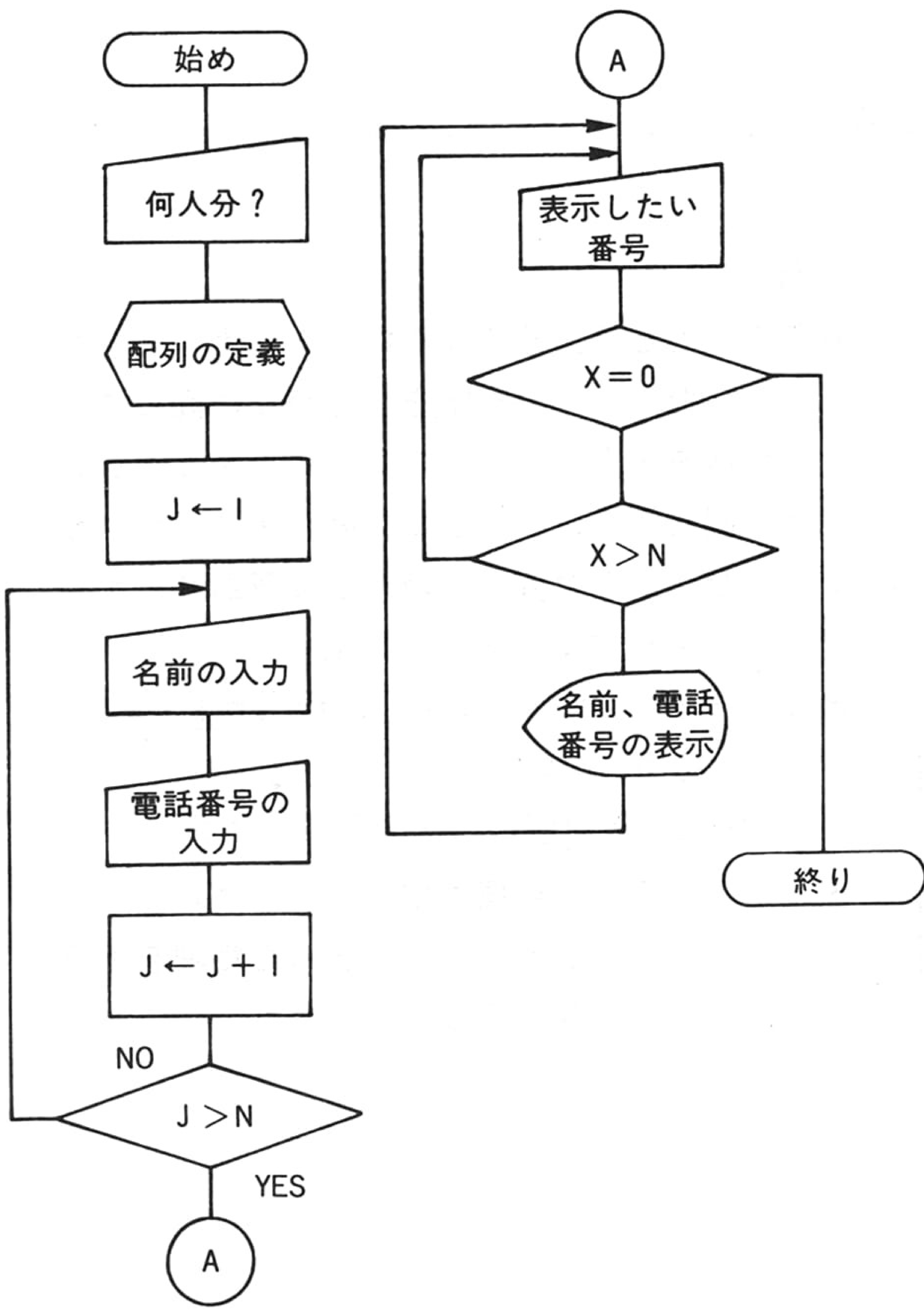
```
10 INPUT"ナンニンフ゛ン ノ テ゛ンワチョウ";N
20 DIM NA$(N),TEL$(N)
30 FOR J=1 TO N
40   PRINT J;"ニンメ"
50   INPUT"ナマエ:";NA$(J)
60   INPUT"テ゛ンワ ハ゛ンコ゛ウ:";TEL$(J)
70   PRINT
80 NEXT J
90 INPUT"ナンニンメ(0)ハ オワリ:";X
100 IF X=0 THEN 150
110 IF X>N THEN 90
120 PRINT  NA$(X);" ノ テ゛ンワハ ";TEL$(X);" テ゛ス"
130 PRINT
140 GOTO 90
150 END
```

このプログラムは、最初に何人分のデータを記憶するかを聞いてきますので、必要な人数を入力します。そのあと、その人数分だけ名前と電話番号の入力をします。

入力が終りますと、何人目のデータを見たいかを聞いてきますので、入力してください。

０を入力すると処理は終了します。

## フローチャート




10行　何人分のデータを入力するか？

20行　配列の宣言

DIM NA＄(N):DIM TEL＄(N)としなくても配列変数を並べて、ひとつのDIM文で２つ以上の配列変数の定義ができます。

30行　FOR文で人数分の入力をします。

40行　何人目の入力かを表示します。

50行　名前の入力

60行　電話番号の入力

70行　１行あける。

80行　NEXT文 くり返し処理を終える。

90行　表示したい人の数字(何人目か)を入力する。

100行　０を入力したら処理を終了する。

110行　範囲のチェック、総人数より大きい指定はできない。

120行　データの表示

130行　１行あける。

140行　GOTO文でループ

150行　処理の終了

### 変数リスト

N：総人数

NA＄(　)：名前を記憶する配列変数

TEL＄(　)：電話番号を記憶する配列変数

J：ループ変数

X：表示したい番号

### (4) 添字（そえじ）

電話メモのプログラムで、データを入力するときにNA$(J)という変数名とTEL$(J)という変数名を使いました。さらにデータを表示したいときにもNA$(X)、TEL$(X)としました。このカッコのなかは、数値でも数値変数でもかまいません。

5番目の人の名前を表示したいときは次のようにします。

```
PRINT  NA$(5)
```

X番目の人の名前を表示したいときは

```
PRINT  NA$(X)
```

とします。

カッコのなかに数値や数値変数を入れて、何番目かを指定します。このカッコのなかの数字を添字といいます。

**メモ**

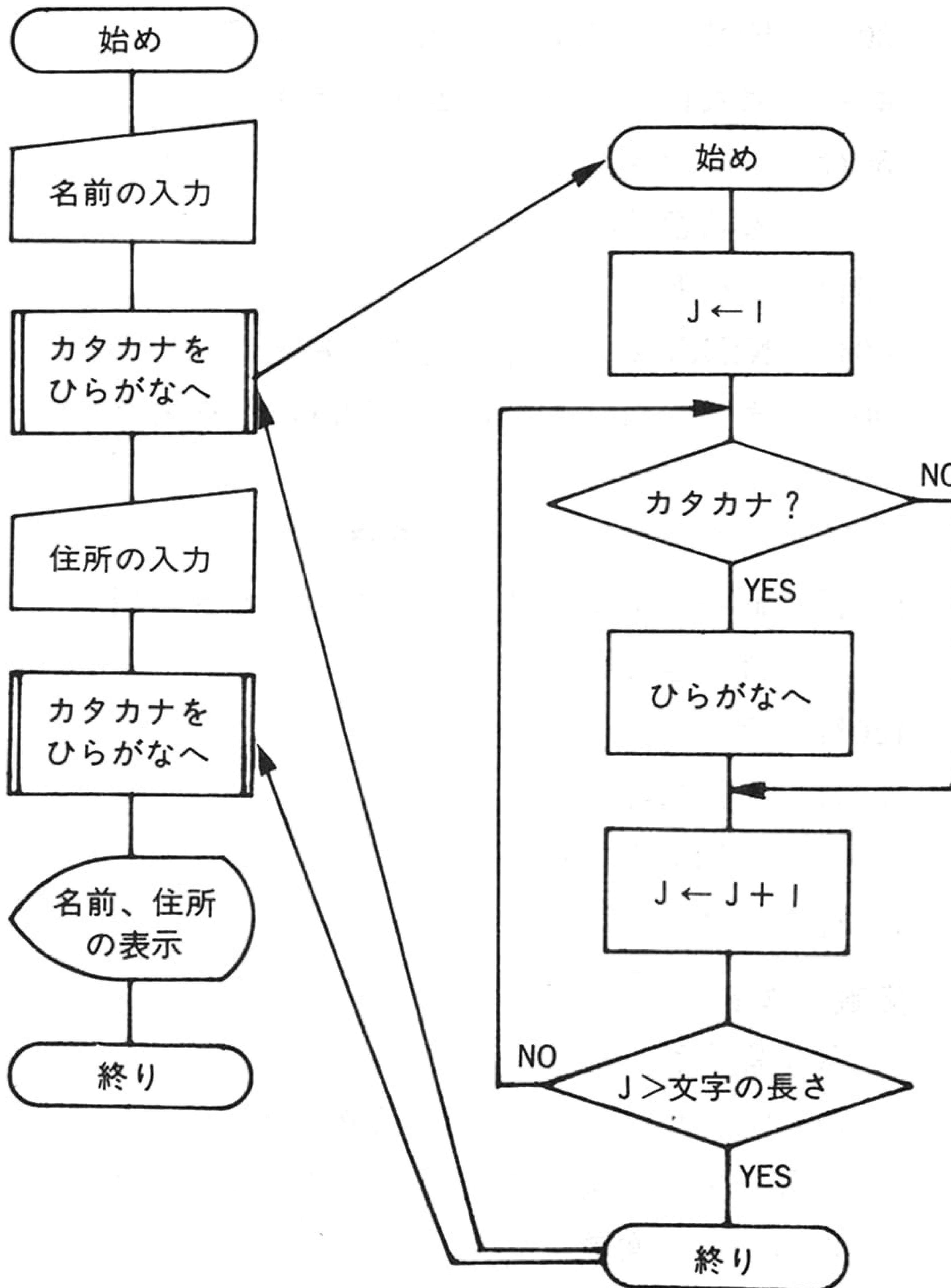
配列変数はカッコのなかが2つの2次元配列の指定もできます。

```
DIM  A$(10, 10)
```

このような指定をすれば、さらにデータの取扱いが簡単になります。

## 4-5 サブルーチンを使おう

プログラムを作っているときに、同じ処理が何回か必要なときがあります。このようなときは、同じ処理をまとめて作っておきます。




〈例〉

名前を入力して、そのなかにカタカナの文字が入っていたら、その文字をひらがなに直します。次に住所を入力して、そのなかにカタカナがあれば、同じようにひらがなに直します。カタカナからひらがなに直す処理が2ヵ所ありますので、これをサブルーチンにします。

名前 | たナカ　タだし | 住所 | トウキョウト |
↓ひらがなに　↓
| たなか　ただし | | とうきょうと |

メインルーチン：処理の中心になるプログラム

サブルーチン：ひとつのまとまった処理をする部分的なプログラム

サブルーチンは、RETURN文を実行すると、GOSUB文のあとからプログラムが継続されます。
次のプログラムは、10→20→100→110→30→40→50→100→110→60→70が実行される順序です。

```
10 A=10
20 GOSUB 100
30 PRINT A
40 A=20
50 GOSUB 100
60 PRINT A
70 END
100 A=A*A
110 RETURN
```

**まとめ**

サブルーチンの実行にはGOSUB文
サブルーチンからもどるにはRETURN文

では、さきほどの例をサブルーチンを使ったときと使わないときにわけて考えましょう。

**サブルーチンを使わないとき**

※ⓐ、ⓑはほぼ同じ処理

```
    10 INPUT"ﾅﾏｴ";NA$
  ┌ 20 FOR J=1 TO LEN(NA$)
  │ 30 CODE=ASC(MID$(NA$,J,1))
  │ 40 IF &HC0<=CODE AND CODE<=&HDD THEN CODE=CODE+&H20
ⓐ │ 50 IF &HA6<=CODE AND CODE<=&HAF THEN CODE=CODE-&H20
  │ 60 IF &HB1<=CODE AND CODE<=&HBF THEN CODE=CODE-&H20
  │ 70 MID$(NA$,J,1)=CHR$(CODE)
  └ 80 NEXT J
    90 INPUT"ｼﾞｭｳｼｮ";AD$
  ┌ 100 FOR J=1 TO LEN(AD$)
  │ 110 CODE=ASC(MID$(AD$,J,1))
  │ 120 IF &HC0<=CODE AND CODE<=&HDD THEN CODE=CODE+&H20
ⓑ │ 130 IF &HA6<=CODE AND CODE<=&HAF THEN CODE=CODE-&H20
  │ 140 IF &HB1<=CODE AND CODE<=&HBF THEN CODE=CODE-&H20
  │ 150 MID$(AD$,J,1)=CHR$(CODE)
  └ 160 NEXT J
    170 PRINT
    180 PRINT"ﾅﾏｴ    :";NA$
    190 PRINT"ｼﾞｭｳｼｮ:";AD$
    200 END
```

### サブルーチンを使ったとき

※サブルーチンのなかでHENKAN$という変数に代入された文字列をひらがなに直していますので、呼び出す前にHENKAN$に変換したい文字列を入れてからGOSUB文を実行します。
また、サブルーチンからもどったときには、HENKAN$の値を使います。このようにサブルーチンのデータの受け渡しに使う変数を引数といいます。

```
10 INPUT"ﾅﾏｴ";NA$
20 HENKAN$=NA$
30 GOSUB 150
40 NA$=HENKAN$
50 INPUT"ｼﾞｭｳｼｮ";AD$
60 HENKAN$=AD$
70 GOSUB 150
80 AD$=HENKAN$
90 PRINT
100 PRINT "ﾅﾏｴ    :";NA$
110 PRINT "ｼﾞｭｳｼｮ:";AD$
120 END
130 REM SUBROUTINE
140 REM HENKAN$ ...ﾋｷｽｳ
150 FOR J=1 TO LEN(HENKAN$)
160 CODE=ASC(MID$(HENKAN$,J,1))
170 IF &HC0<=CODE AND CODE<=&HDD THEN CODE=CODE+&H20
180 IF &HA6<=CODE AND CODE<=&HAF THEN CODE=CODE-&H20
190 IF &HB1<=CODE AND CODE<=&HBF THEN CODE=CODE-&H20
200 MID$(HENKAN$,J,1)=CHR$(CODE)
210 NEXT J
220 RETURN
```



**●サブルーチンを使うときの注意点**

①サブルーチンのなかではCLEAR文は使えません。RETURN文が実行できなくなるからです。
②サブルーチン内で使う変数は、メインルーチンに影響しない変数にしましょう。
③サブルーチンはひとつのまとまったプログラムですからサブルーチンの先頭に見出し（コメント）をつけるとわかりやすくなります。

## 4-6 美しいプログラム

プログラムは、変数のつけ方もある程度は自由です。また、命令と命令の間に空白（スペース）を入れることもできます。行番号も何行からはじめてもかまいません。このようなことから、プログラムは作る人によって見やすいプログラムになったり、見づらいプログラムになったりします。見やすいプログラムというのは、プログラムを見ただけで、何をしているのかすぐに理解できるものです。
ここでは、プログラムを美しく作るヒントを説明しましょう。

### (1) コメントを多くつける

コメントはREM文またはシングルクォーテーション(')のあとに自由に書くことができます。
行全体をコメントすることも、1行の途中からコメントにすることもできます。

### (2) 段さげを利用する

行番号と命令の間の空白（スペース）は、ひとつとは限りません。必要に応じて空白をふやすこともできます。
FOR~NEXT文のときの対応をつけるのに使うと、FOR~NEXT間がどこまでかがよくわかります。

### (3) 命令と命令の間などに空白を入れる

BASICの1行のなかには、命令文もあれば、変数も数値もあります。変数と命令の間に1文字空白をあけるだけで、プログラムは見やすいものになります。

```
20 CLS
30 FORJ=1TO10
40 FORK=1TO10
50 L=J*K
60 PRINT L
70 NEXT K
80 NEXT J
90 END
```

あまりよくない例

```
10 REM ｶﾞﾒﾝ ｦ ｹｽ
20 CLS
30 FOR J=1 TO 10          REM J=1 ｶﾗ 10
40   FOR K=1 TO 10        REM K=1 ｶﾗ 10
50     L=J*K              REM L ﾉ ｹｲｻﾝ
60     PRINT L            REM L ﾉ ﾋｮｳｼﾞ
70   NEXT K
80 NEXT J
90 END                    REM ｵﾜﾘ
```

美しいプログラム例

ただし、美しさだけにとらわれると、メモリーの空き領域が少なくなります。

# 第5章 プログラムを作ろう

ここでは、MX-101の特長をフルに発揮したプログラムを例に、プログラムの作り方、命令の使い方を説明していきます。プログラムを入力するときは必ずNEW ↵ を実行してから行なってください。

## 5-1 必ずマスターしたい命令とその使い方

すでに説明したNEW、LIST、RUN命令以外にプログラミングに欠かせない重要な命令を紹介します。

### (1) PRINT命令

PRINT命令は、データ（文字列）などを画面に表示する命令で、次に続くメッセージや変数の内容、計算結果を表示します。
この命令を使っていろいろな計算をしてみましょう。まず簡単なたし算をして結果を表示してみます。

例）
```
10 PRINT 5+10
```
このプログラムは5＋10の結果を表示するものです。
実行は F5 か R U N ↵ で行ないます。

```
10 PRINT 5+10
run
 15
Ok
```

では次のプログラムはどうでしょう。

例）
```
10 A=5+10
20 PRINT A
```
5＋10の結果を変数Aに覚えさせて変数Aの内容を表示させます。

```
10 A=5+10
20 PRINT A
run
 15
Ok
```

次に四則演算を行なってみましょう。BASICでのかけ算は＊(アスタリスク)、わり算は ／ (スラッシュ)を使います。

例）
```
10 A=7
20 B=8
30 PRINTA+B
40 PRINTA-B
50 PRINTA*B
60 PRINTA/B
```



```
run
 15
-1
 56
 .875
Ok
```

さて、これまでは結果だけを表示しましたが、結果の前にコメントをつけることもできます。

例)
```
10 A=5
20 B=10
30 PRINT"A+B=";A+B
```

```
run
A+B= 15
Ok
```

30行のようにダブルクォーテーション（"）でかこまれた文字列はそのままの形で表示します。
A＋B＝を表示したあとにA＋Bの結果が表示されます。

## ■PRINT文の書き方

PRINT文では、文字列や結果などの間隔をあけて表示したり、つめて表示したりする方法があります。これはカンマ(,)とセミコロン(;)を使用します。この方法を知っておくとプログラム作りにとても便利です。

①カンマ（,）を使う

,で区切ると変数AとBの数値は、一定の間隔に区切られます。その間隔は、符号(＋(空白)、－)を含めて14桁に区切って表示します。,を2つ続けると数値は2行にわたり表示されます。
,をとると新しい変数ABと受けとります。ABには何の定義もしていませんのでコンピュータは0を表示します。

```
10 A=5
20 B=10
30 PRINT A,B
run
␣5            ␣10
```

14桁

符号(プラスのときは空白、マイナスのときは－を表示)

②セミコロン（;）を使う

;で区切ったときは、変数AとBの数値の間に2つの空白があきますが、数値の前につく空白は符号のための空白で、数値のあとにもうひとつ空白がつきます。

```
10 A=5
20 B=10
30 PRINT A;B
run
␣5␣␣10␣
```

符号　空白

③2つのPRINT文で使う

2つのPRINT文のデータに最後に,も;もつけないときは改行されて表示されます。

```
10 A=5
20 B=10
30 PRINT A;
40 PRINT B
run
 5  10
Ok
```

（セミコロンをつけたとき）

```
10 A=5
20 B=10
30 PRINT A
40 PRINT B
run
 5
 10
Ok
```

（セミコロンをつけないとき）

④カンマ(,)、セミコロン(;)を使わない

PRINT文で表示したいデータを ,と;を使わないで変数を続けて書くこともできます。それは変数に属性文字($、%、!、#)があるときです。

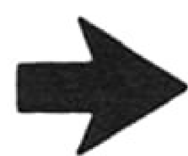
例)
```
10 A$="CASIO"
20 B$="MSX"
30 PRINT A$,B$
40 PRINT A$;B$
50 PRINT A$B$
60 A#=-10
70 B#=-20
80 PRINT A#,B#
90 PRINT A#;B#
100 PRINT A#B#
```

```
CASIO          MSX
CASIOMSX
CASIOMSX
-10            -20
-10 -20
-10 -20
```

⑤文字や記号はきれいに間をあける

文字や記号は、数値とちがって符号+(プラス)、-(マイナス)はつきません。文字や記号はセミコロン(;)で区切ると1文字分もあけずに表示します。もし間をあけたいときは、空白を〝␣〟のようにダブルクォーテーションでかこみます。

```
10 PRINT"CASIO";"MSX"
run
CASIOMSX
```



```
10 PRINT"CASIO";"  ";"MSX"
run
CASIO  MSX
```

## (2) INPUT命令

INPUT命令は、キーボードからデータを入力する命令です。では次のプログラムを入力してみましょう。

```
10 INPUT A
20 PRINT A
30 END
```

10行のINPUT　Aは、キーボードから入力したデータを変数Aに入れなさいという命令です。20行はそのデータを表示します。このプログラムを実行するとMSXは〝?〟を表示してデータの入力待ちとなります。

右の実行例はデータに432を入力し、[↵]を押した例です。ここでもデータを入力したあと必ず[↵]キーを押します。

[4][3][2][↵]

```
10 INPUT A
20 PRINT A
30 END
Ok
run
? 432 ←入力する
 432
Ok
```

数字以外の文字を入力すると？Redo from start が表示されますが、再びデータを入力することはできます。

```
run
? ABC ←文字を入れる
?Redo from start
? 123 ←今度は数字を入れた
 123
Ok
```

INPUT文は、メッセージを表示したあとにデータ入力をすることができます。INPUTのあとにダブルクォーテーション（"）でメッセージを囲みセミコロン（；）で区切ります。

```
10 INPUT"A=";A
20 PRINT A
30 END
run
A=? 10
 10
Ok
```

ひとつのINPUT文で複数のデータを入力することもできます。このときの変数はカンマ（，）で区切ります。データの入力は、カンマ（，）で区切ってやる方法と、ひとつひとつ入力する方法とがありますが、ひとつひとつ入力するときは、2つめの入力のときに〝？？〞が表示されます。

```
10 INPUT A,B
20 PRINT A+B
30 END
run
? 10,20←
 30       [1][0][,][2][0][↵]とした
Ok
```

```
run
? 10 ←[1][0][↵]とした
?? 20←[2][0][↵]とした
 30
Ok
```

### (3) GOTO文

プログラムは行番号の小さい順に実行しますが、GOTO文を使うと、自由に行をとばすことができます。GOTO文は、GO(空白)TOでもGOTOとしても、どちらでもかまいません。GOTOのあとには、これからとんでいく行番号をつけます。

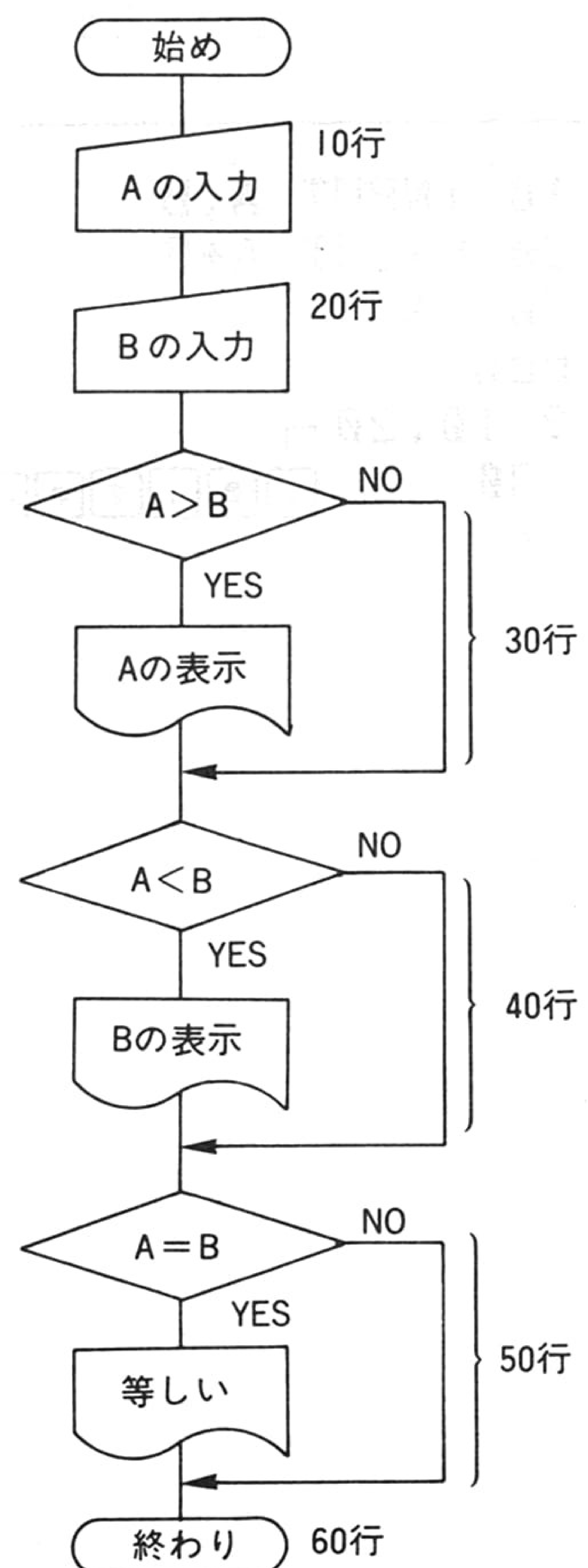
```
10 PRINT "コンニチハ?"
20 GOTO 50
30 PRINT "3"
40 GOTO 70
50 PRINT "1"
60 GOTO 90
70 PRINT "4"
80 END
90 PRINT "2"
100 GOTO30
```

このプログラムを実行するとどうなるか、ちょっと考えてみてください。
まずRUNすると10行を実行し、画面には「コンニチワ？」と表示されます。次に20行でGOTO 50を実行し、50行にとびます。50行では「1」を表示します。その次はGOTO 90を実行します。
そのあとは……考えてみてください。

### (4) IF文

これは、いろいろな条件の判断を行なう命令です。「もし～ならば～をする」というようなことをするわけです。

```
10 INPUT"A=";A
20 INPUT"B=";B
30 IF A>B THEN PRINT A;"カ゛ オオキイ"
40 IF A<B THEN PRINT B;"カ゛ オオキイ"
50 IF A=B THEN PRINT "ヒトシイ"
60 END
```



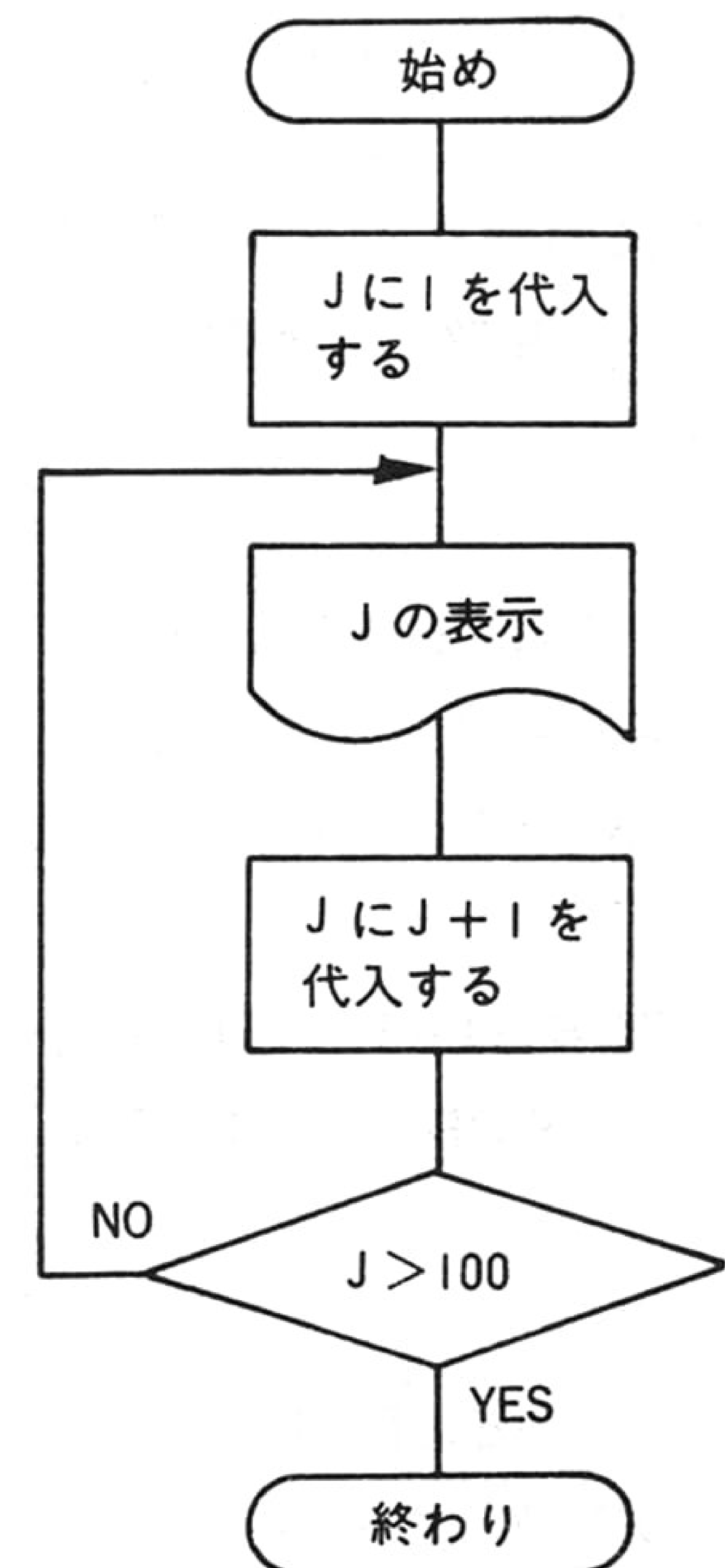

このプログラムはAとBを入力し、AとBの値を比較するプログラムです。30行の〝A＞B〟という条件が成立すれば、THENのあとの命令を実行します。以下40行、50行と同じ内容です。〝A＞B〟を条件式といいます。
またIF文はIF①THEN②ELSE③のような使い方もできます。これは「①(条件式)がYESなら②を、NOならば③を実行しなさい」という形になります。
THENまたはELSEのあとにGOTO行番号を書くときはGOTOを省略することができます。
たとえば「10 IF A＝B THEN 200」という形です。

## (5) FOR～NEXT文

くり返しを行なう命令です。

FORとNEXTの間にはさまれた命令を実行します。たとえば、前に学習したIF文を使って1から100まで表示しようとすると、どうなるでしょう。

```
10 J=1
20 PRINT J
30 J=J+1
40 IF J>100 THEN 50 ELSE 20
50 END
```




このプログラムでは、変数「J」の初期値をセットする部分が10行、Jの値に1プラスするのが30行、Jが100よりも大きいかどうか判断するのが40行になります。

FOR～NEXT文は、このようなくり返し処理を行なうのにとても便利な命令です。1から100までの表示を行なうプログラムをFOR～NEXT文で作成すると次のようになります。

```
10 FOR J=1 TO 100
20 PRINT J
30 NEXT J
40 END
```

10行のFOR文は「変数Jが初期値1から最終値100まで」という意味です。30行のNEXT文は、10行のFOR文と対応するもので、このNEXT文の前までが実行されます。

では、数字を1、3、5、7……と、1つおきに表示したいときには、どうすればいいでしょうか？
FOR文の書き方は

[行番号]FOR[変数]＝[初期値]TO[最終値]STEP[増分]

となります。このようにSTEP以降を省略すると、増分は1になります。また増分にマイナス（－）を使うこともできます。
そこで、プログラムは

10 FOR J＝1 TO 100 STEP 2

です。前のページのプログラムにSTEP 2を追加して実行してください。
FOR文の終りであるNEXT文には、NEXTのあとにFOR文で使った変数名を書きます。
このとき変数名を省略することもできますが、FOR文との対応がわかりにくくなりますので、省略しない方がいいでしょう。

FOR～NEXT文のなかに、さらにFOR～NEXT文を入れることもできます。この方法を「ネスティング」(入れ子) と呼びます。ただし、このときFORとNEXTの対応がまじわることは許されません。

```
FOR J=1  TO  10 ─────┐
   FOR  K=1  TO  10─┐ │
          ∫         │対│対
                    │応│応
   NEXT  K ─────────┘ │
FOR J ────────────────┘
```

```
FOR J=1  TO  10 ─────┐
   FOR  K=1  TO  10──┼──┐
          ∫          │  │
   NEXT  J ──────────┘  │
NEXT  K ────────────────┘
```

まじわっているので許されません。
"NEXT without FOR" というエラーになります。

```
10 SCREEN 0
20 WIDTH 40
30 FOR J=1 TO 9 ─────┐
40   FOR K=1 TO 9 ─┐ │
50     PRINT J*K;  │ │
60   NEXT K ───────┘ │
70   PRINT           │
80 NEXT J ───────────┘
90 END
```

NEXT文が2つ以上つながっているときは、1つのNEXT文で変数をつなぐことができます。たとえばNEXT KのあとにNEXT Jが続くときは、NEXT K、Jという具合です。当然NEXT J、Kではいけません。必ずループの内側にある方が左側にきます。
FOR～NEXT文の間にもう1つのFOR～NEXT文を入れた例を次に示します。これは九九（くく）の表を表示するプログラムです。

### (7) GOSUB～RETURN

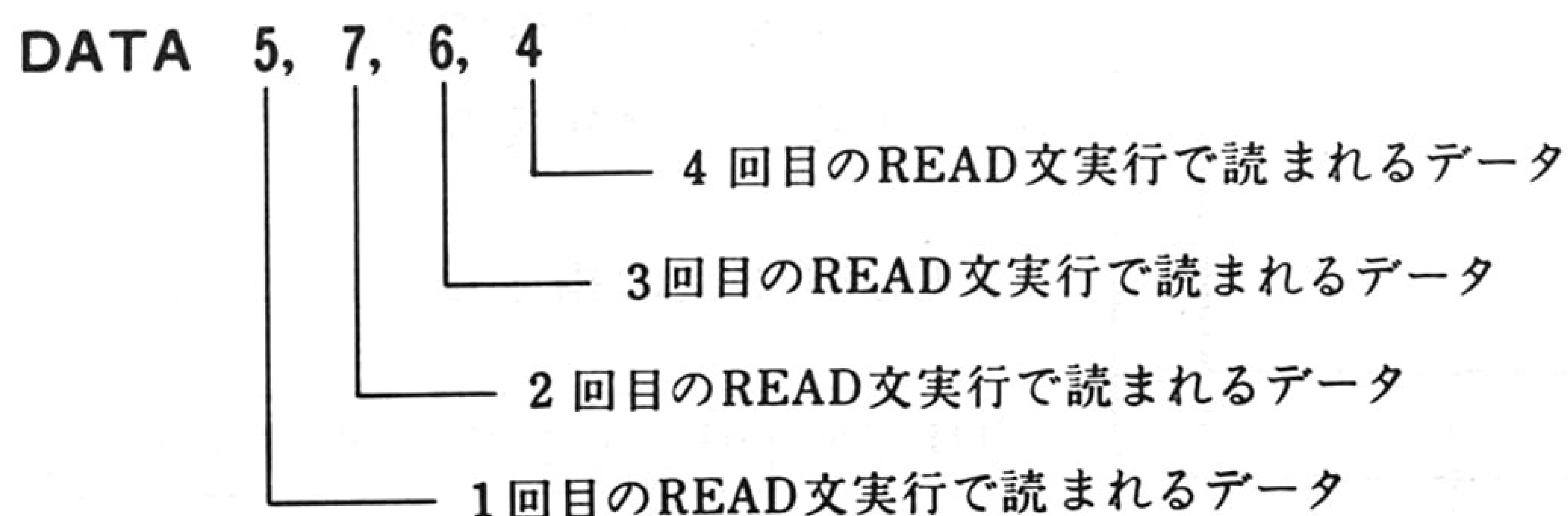
サブルーチンを呼び出す命令がGOSUB文で、サブルーチンからもどる命令が、RETURN文です。サブルーチンについては、〝4－5サブルーチンを使おう〞で詳しく説明しますから、ここでは簡単に書き方だけを説明します。GOSUB文は、GOSUB[行番号]になります。[行番号]は必ず書いてください。このとき、まちがって存在しない行番号を指定すると〝Undefined line number〞エラーになります。RETURN文は、RETURN[行番号]と書きますが、[行番号]は省略することができます。省略したときは、GOSUB文の次の命令を実行し、[行番号]をつけたときは、その行番号から実行します。

### (8) READ文とDATA文およびRESTORE文

プログラム中で使う値をプログラム中に書き込んで、必要に応じて読み出すことができます。これを行う命令がREAD文で、データの設定を行なうのがDATA文です。ここでREAD文で読み込む変数の型と、読み出されるデータの型は同じでなければなりません。

●READ文の書式
　変数がひとつのとき：READ変数名
　変数が複数のとき　：READ変数名，変数名…

DATAの個数以上にデータを読み込もうとしたときは〝Out of DATA〞エラーになります。また変数の型が異なるときは〝Type mismatch〞エラーになります。
DATA文はプログラム中のどこに存在してもかまいません。書式は次の通りです。
　　DATA　定数，定数……
またDATA文のデータはREAD文で読まれるごとに、１つずつデータを与えます。




```
10 FOR J=1 TO 3
20   READ A
30   PRINT A
40 NEXT J
50 END
60 DATA 5,4,7
70 DATA 3,9,6
```

このプログラムは、READ文がFOR～NEXT文で3回くり返されます。READ文に対応するDATA文は、プログラム全体のDATA文の早い行番号のデータから１つずつ読み込まれます。このプログラムを実行すると３回読みこまれるので60行の5、4、7が表示されることになります。

RESTORE文で読み込まれるDATA文を指定できます。

```
5 RESTORE 70
10 FOR J=1 TO 3
20    READ A
30    PRINT A
40 NEXT J
50 END
60 DATA 5,4,7
70 DATA 3,9,6
```

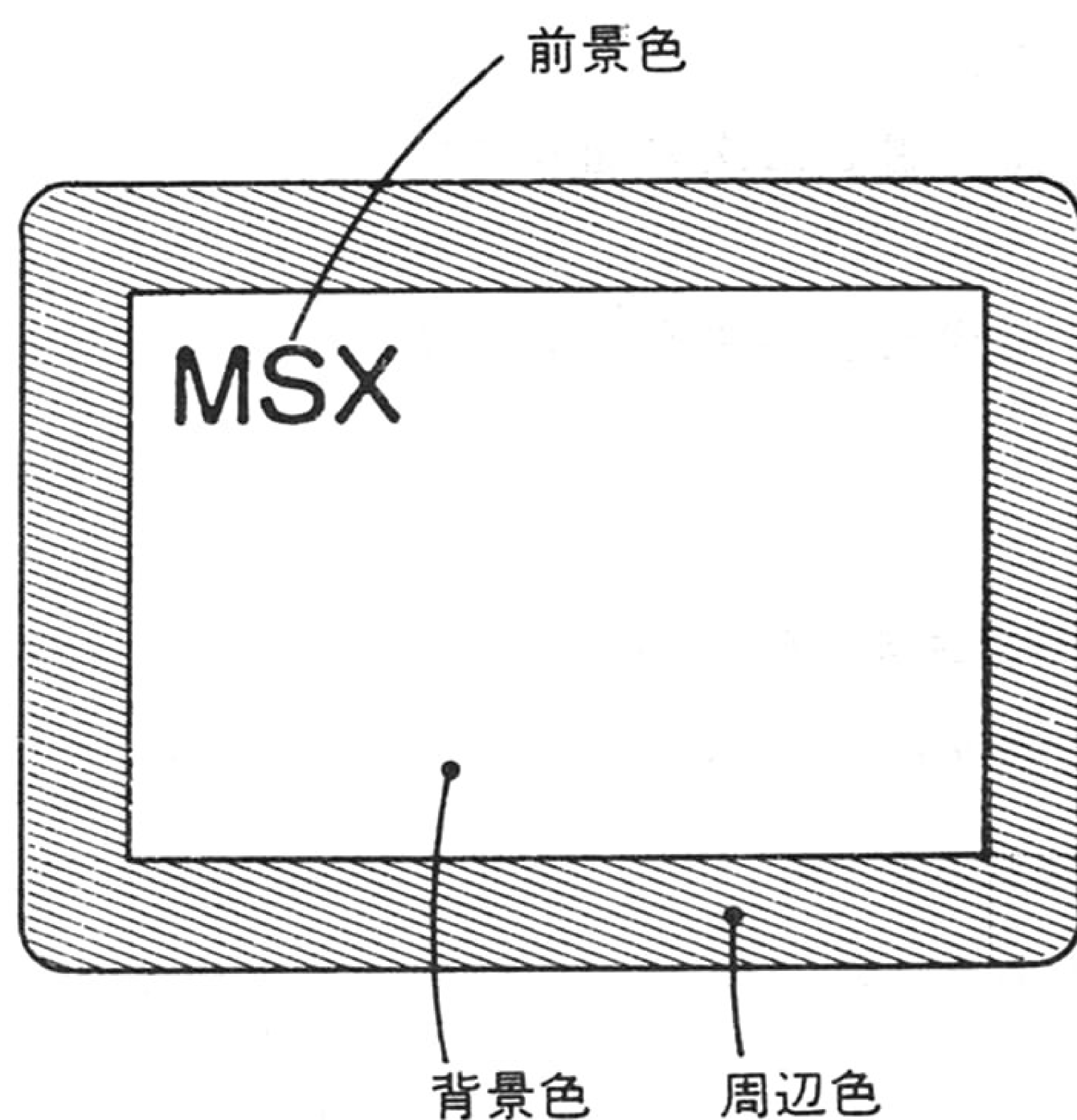
このプログラムは5行目のRESTORE文で、読み込むDATA文を70行に指定しています。これを実行すると3、9、6が表示されます。

## 5-2 MSXをもっと楽しむ命令（グラフィック）

いままで学習した命令だけでも、充分にプログラムを作ることができますが、ここでは、もっと楽しいプログラムを作るため、さらにいろいろな命令を説明していきます。

### (1) COLOR文

MX-101はテレビに16色の色を出すことができますが、この色の指定を行なう命令が、COLOR文です。16の色はそれぞれ色コードで対応し、色の指定はテレビ画面上の3つの部分をそれぞれ指定することができます。




- 前景色は文字の色、グラフィック画面での線や点の色です。
- 背景色は文字を書く画面の色、グラフィック画面の地の色です。
- 周辺色は背景の外わくの色です。

COLOR文の指定は次のようになります。

COLOR [前景色番号], [背景色番号], [周辺色番号]

色コード（カラーコード）と実際の色は次のようになります。

| | |
|---|---|
| 0＝透明（周辺色と同じ色） | 8＝赤 |
| 1＝黒 | 9＝明るい赤 |
| 2＝緑 | 10＝黄 |
| 3＝明るい緑 | 11＝明るい黄 |
| 4＝暗い青 | 12＝暗い緑 |
| 5＝明るい青 | 13＝紫 |
| 6＝暗い赤 | 14＝灰 |
| 7＝水色 | 15＝白 |

```
5 CLS
10 DEFINT A-Z
20 PRINT "ｼｭｳﾍﾝｼｮｸｦ ｶｴﾏｽ"
30 FOR J=0 TO 15
40 GOSUB 270
50 COLOR ,,J
60 GOSUB 230
70 NEXT J
80 PRINT "ﾊｲｹｲｼｮｸｦ ｶｴﾏｽ"
90 FOR J=0 TO 15
100 GOSUB 270
110 COLOR ,J
120 GOSUB 230
130 NEXT J
140 PRINT "ｾﾞﾝｹｲｼｮｸｦ ｶｴﾏｽ"
150 FOR J=0 TO 15
160 GOSUB 270
170 COLOR J
180 GOSUB 230
190 NEXT J
200 COLOR 15,4,7
210 END
220 REM ｼﾞｶﾝﾏﾁ
230 FOR T=1 TO 900
240 NEXT T
250 RETURN
260 REM ｲﾛﾋｮｳｼﾞ
270 PRINT "ｶﾗｰ";J
280 RETURN
```

このプログラムは、前景色、背景色、周辺色を変化させるプログラムです。
例えば前景色だけを変えるときはCOLOR (色コード) とし、背景色、周辺色は省略できます。
同様に、周辺色だけを変えるときはCOLOR,, (色コード) とします。

## (2)SCREEN文 (画面の種類の選択)

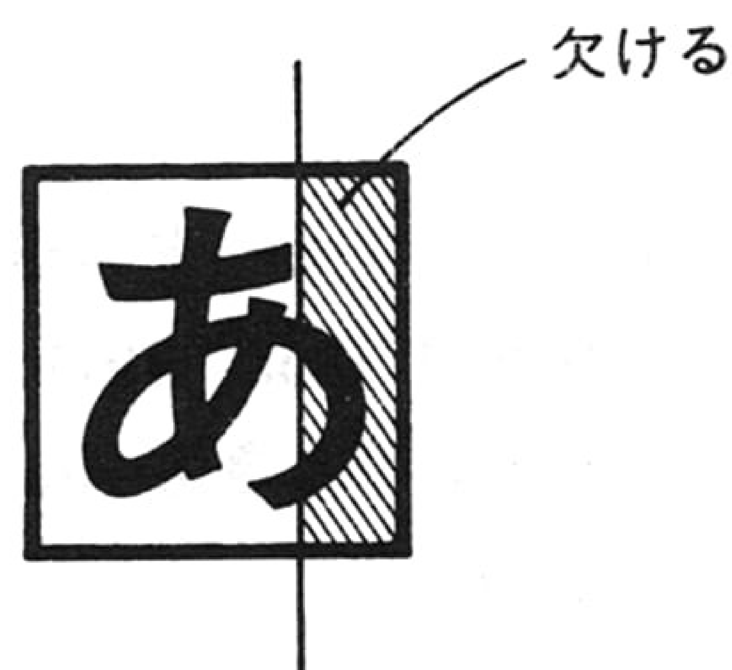
MSXでは0～3まで4種類の画面モードをもっています。

0＝最大40文字×24行で構成されるテキストモード
1＝最大32文字×24行で構成されるテキストモード
2＝高解像度グラフィック(256×192ドットで構成される)モード
3＝マルチカラーモード

このなかでテキストモードと呼ばれる画面の状態のとき、プログラムの入力を行なうことができます。ところで、これまでのプログラムはテキストモードのプログラムでしたが、画面に線を描くことができるのは、高解像度グラフィックモードかマルチカラーモードです。

**●最大40文字×24行テキストモード** (SCREEN 0)

このモードでは、前景色の指定と背景色の指定だけが有効になります。画面上に多くの文字が表示できますが、1つの文字は、横6ドット、(6つの点)で表示されるため、ひらがな（横8ドットで構成される文字）は少し欠けて表示されます。また、あとで説明されるスプライトの表示もできません。



**●最大32文字×24行テキストモード**(SCREEN 1)

このモードでは、文字を横方向に最大32文字表示（WIDTH文 73 P参照）することができます。またスプライトを使うこともできます。MX-101に電源を入れたときのモードはSCREEN 1です。色は、周辺色、背景色、前景色（文字の色）の3色が画面に表示されます。ただしスプライトは自由に16色指定できます。

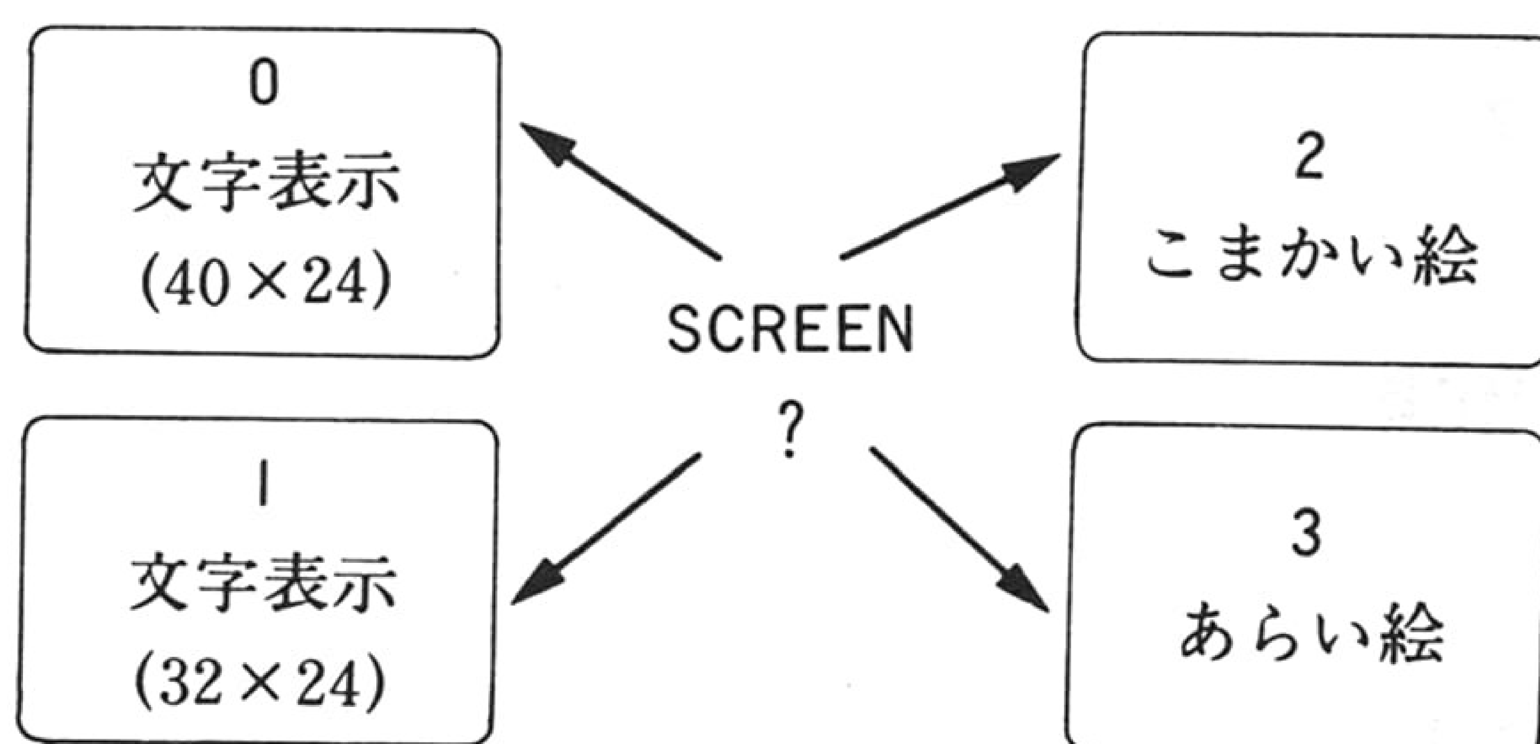
**●高解像度グラフィックモード**(SCREEN 2)

絵を表示するときにこのモードを使います。このモードは横256ドット、たて192ドットの点を表示できます。

**●マルチカラーモード**(SCREEN 3)

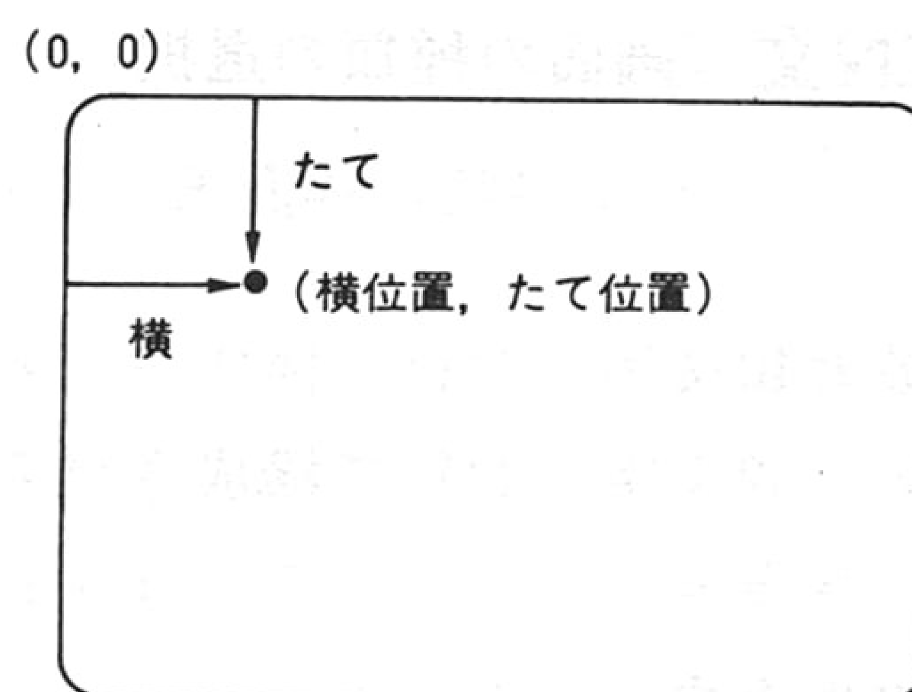
このモードは、横4ドット、たて4ドットをひとまとめ（1ブロック）として表示するモードです。画面全体のブロックは、横64、たて48になります。

各ブロックごとに点を表示することができますが、点の大きさは、高解像度グラフィックモードと比較すると、たて、横、各4倍の大きさになります。




**●座標の指定**

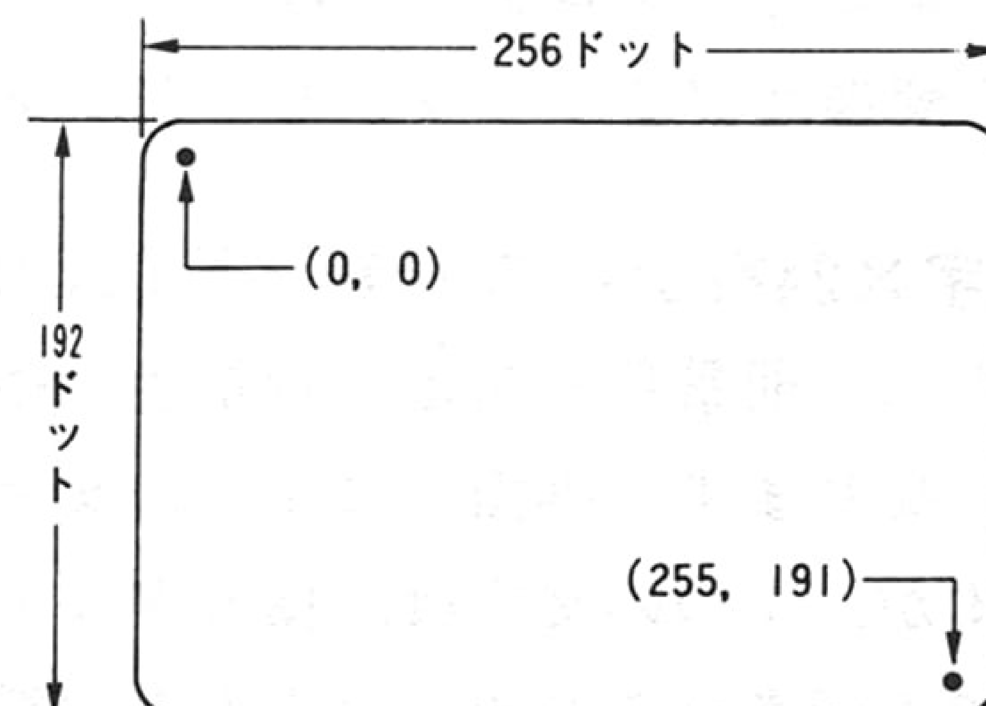
画面上に文字を表示したり、グラフィック点を表示するには、その位置を指定しなければなりません。この位置指定は座標で行ないますが、座標は横位置、たて位置を指定します。




テキストモードで文字を表示するときはLOCATE文で、文字の表示位置を指定することができます。画面の左上の座標が(0,0)となり、右下が(横方向の表示できる文字数−1,23)となります。たとえば、5行目の10文字目のところに文字を表示するときは

LOCATE9,4：PRINT"ABC"

とします。




高解像度グラフィックモードの座標指定は左上が（0, 0）、右下が（255, 191）となります。マルチカラーモードのときも（0, 0）～（255, 191）で表わします。

### (3) WIDTH文

テキスト画面で1行に表示できる最大文字数を決めます。

SCREEN　0のとき――1～40字

SCREEN　1のとき――1～32字

ただし、表示文字数を大きくすると、テレビによっては左側の文字が見えなくなることがあります。

WIDTH文の指定は次の通りです。

WIDTH 最大文字数

### (4) LOCATE文

INPUT文やPRINT文を実行するときの位置を指定します。

これは次のように指定します。

LOCATE X座標, Y座標

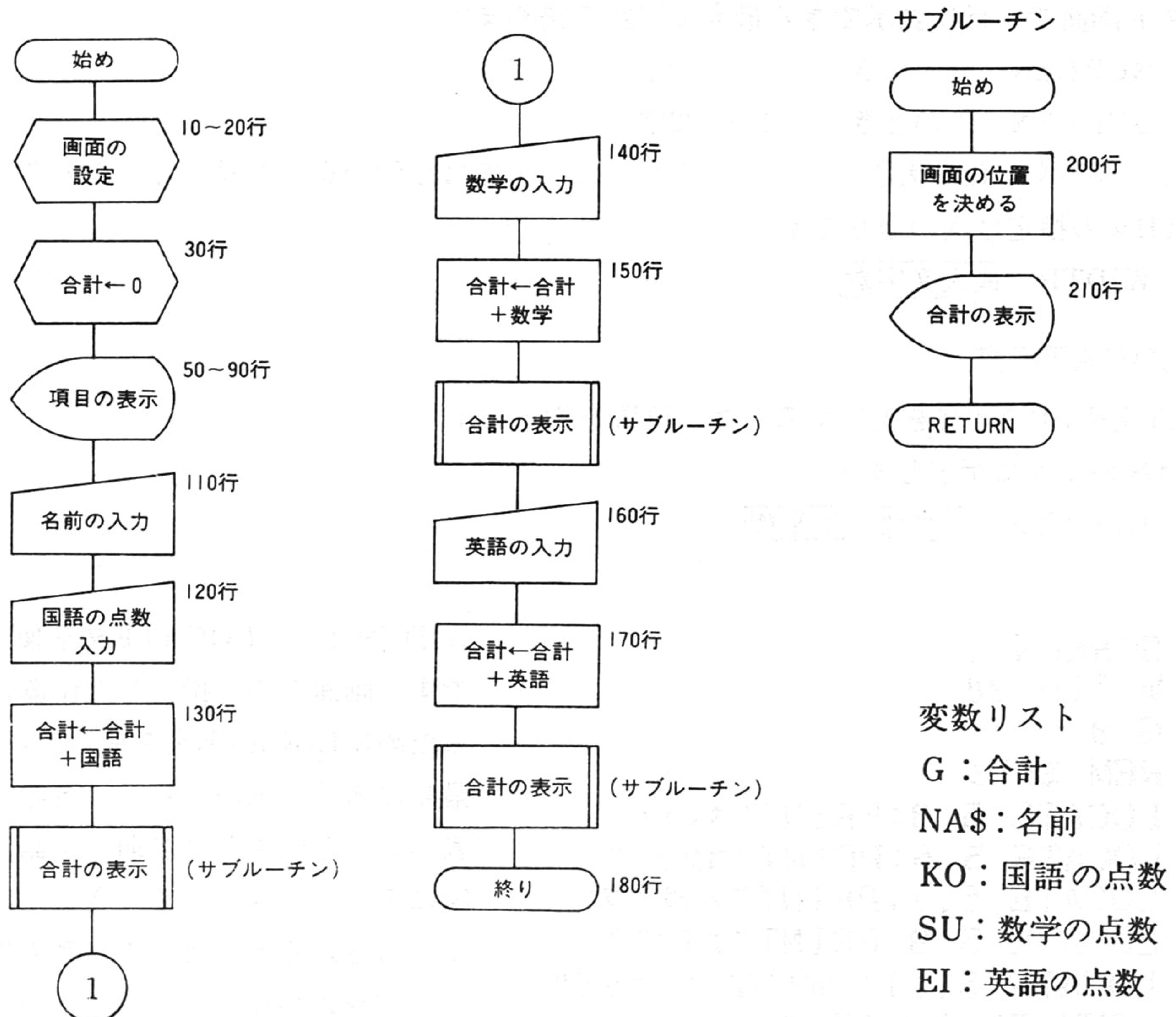
```
10 SCREEN 1
20 WIDTH 28
30 G=0
40 REM ﾐﾀﾞｼ
50 LOCATE 5,3:PRINT"ﾅﾏｴ"
60 LOCATE 5,5:PRINT"ｺｸｺﾞ"
70 LOCATE 5,7:PRINT"ｽｳｶﾞｸ"
80 LOCATE 5,9:PRINT"ｴｲｺﾞ"
90 LOCATE 5,11:PRINT"ｺﾞｳｹｲ"
100 REM ﾃﾞｰﾀﾆｭｳﾘｮｸ
110 LOCATE 13,3:INPUT NA$
120 LOCATE 13,5:INPUT KO
130 G=G+KO:GOSUB 200
140 LOCATE 13,7:INPUT SU
150 G=G+SU:GOSUB 200
160 LOCATE 13,9:INPUT EI
170 G=G+EI:GOSUB 200
180 END
190 REM ｺﾞｳｹｲﾉﾋｮｳｼﾞ
200 LOCATE 14,11:PRINT G
210 RETURN
```

WIDTH文と、LOCATE文を使ったプログラムです。画面上の、指定した位置に文字を表示するためにLOCATE文を使っています。

最初に名前、国語、数学、英語、合計の文字を表示し、そのあとに名前、国語、数学、英語の順にデータを入力していきます。点数のデータが入力されると、合計の点数も増えていきます。

プログラムのなかで使われているコロン（：）は、1行に複数の命令を並べるときの区切りに使いますが、2つ以上の命令が並んでいる状態をマルチステートメントと呼びます。

**フローチャート**




## (5) PSET文

PSET文を使って画面上に点を表示してみましょう。このPSET文は、グラフィック命令なので、必ずSCREEN 2かSCREEN 3にしてから実行します。テキストモードで実行すると、〝Illegal function call〃 エラーとなります。

PSET文は次のように指定します。

PSET（X座標，Y座標），色コード

キーボードから指定した位置に指定した色で点を表示するプログラムです。

```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 INPUT"COLOR=";C
40 COLOR 15,1,1
50 SCREEN 2
60 PSET(X,Y),C
70 GOTO 70
```

10～30行　それぞれのデータを入力します。

40行　COLOR文で画面の背景を黒にします。

50行　グラフィックモードにします。

60行　点を表示します。

70行　プログラムを70行で止めるために、GOTO文を実行しています。

```
run
X=? 128
Y=? 96
COLOR=? 10
```

（画面の中央に黄色い点を表示します）

入力範囲

Xは、0～255

Yは、0～191

Cは、0～15

プログラムが終了するとテキスト画面にもどってしまい、せっかく表示した点が消えてしまいますので、70行目はGOTO70として、いつまでも70行を実行するようにしたわけです。CTRL→STOPで終了します。
PSET文で色コードを指定しないと、COLOR文で指定した前景色になります。

## (6) PRESET文

PRESET文はPSET文とは逆に、指定した座標の位置の点を消します。
この文の指定は次の通り。

PRESET（X座標, Y座標）

PSET文ではCOLOR文で指定した前景色で点を表示しますが、PRESET文は、指定した座標位置に背景色で点を表示します。「背景色で点を表示する……」ということは、つまり点を消すことになるわけです。
PRESET文で色コードを指定するとPSET文で色を指定したのと同じ働きをします。

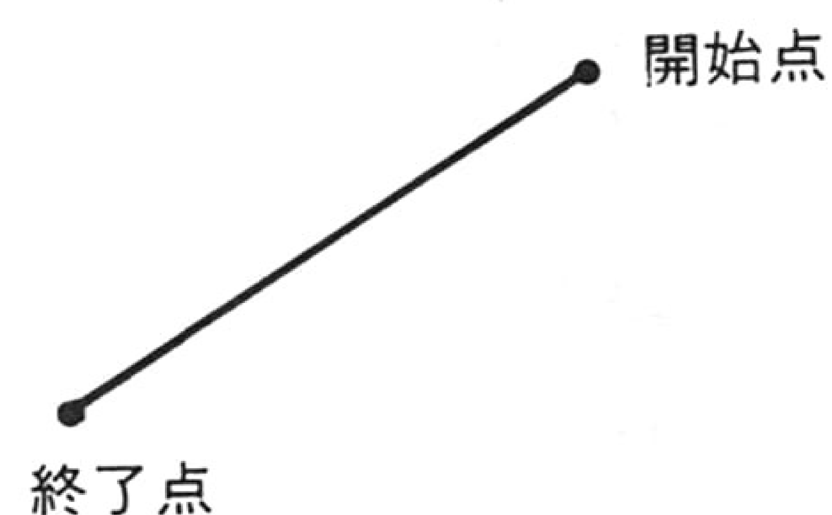
**まとめ**

PSET（X座標，Y座標）―――― 前景色で点を表示
PRESET（X座標，Y座標）―――― 点を消す（背景色で点を表示）
PSET（X座標，Y座標），色コード―― 指定した色コードに対応する色で点を表示する
PRESET（X座標，Y座標），色コード―― 指定した色コードに対応する色で点を表示する

## (7) LINE文①

LINE文は、その名の通り線を引く命令です。線を引くには、引き始めの点と引き終りの点があってはじめて線を引くことができます。また、BASICでも２つの点を指定します。

LINE（開始点）－（終了点），色コード

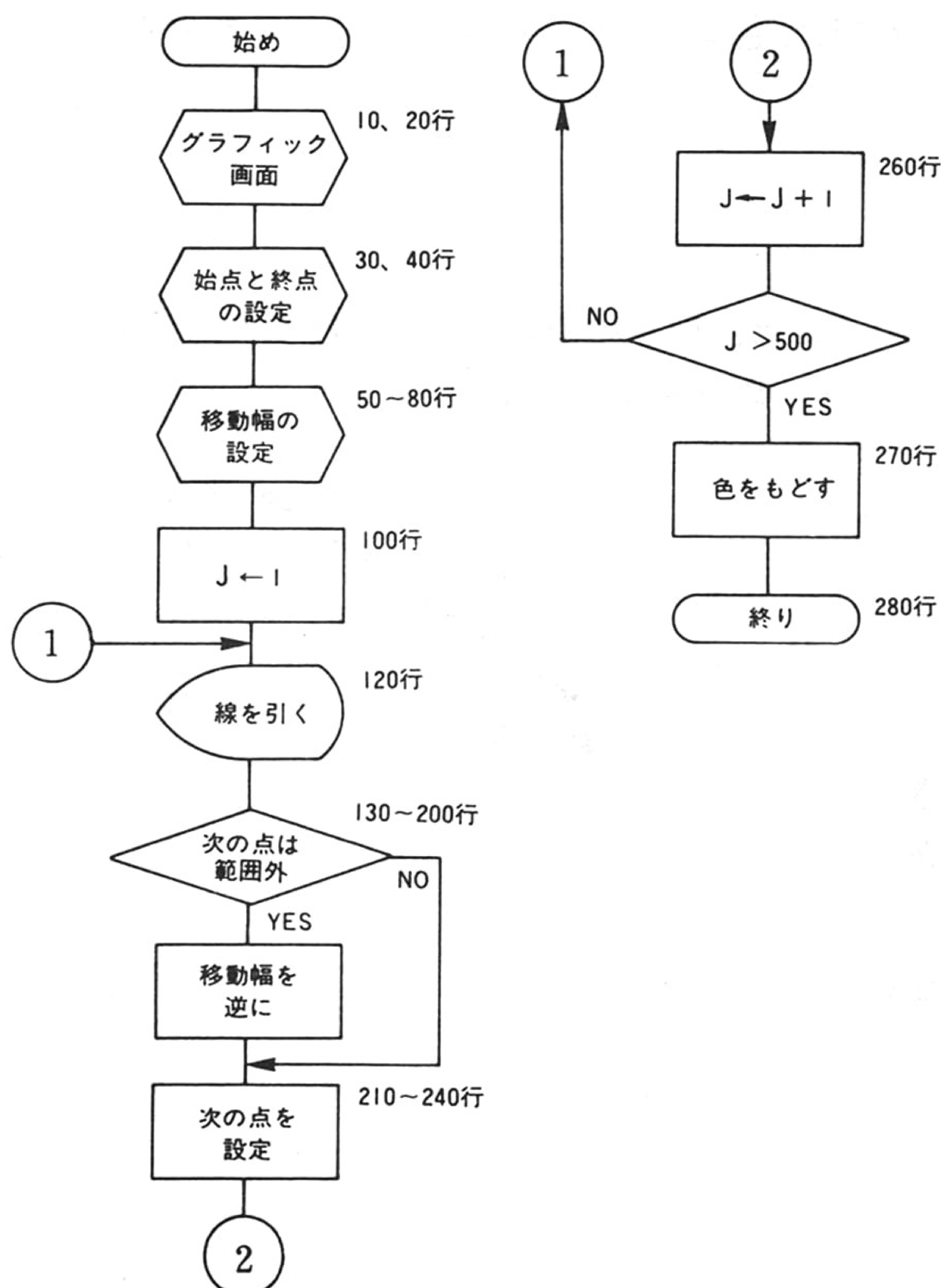



開始点、終了点ともに座標で表わします。このとき、色コードを省略するとCOLOR文で指定した前景色になります。
なおLINE文もSCREEN 2 またはSCREEN 3 で使います。

```
10 COLOR 15,1,1
20 SCREEN 2
30 X1=0:Y1=0
40 X2=10:Y2=10
50 DX=5
60 DY=3
70 EX=3
80 EY=4
90 REM
100 FOR J=1 TO 500
110 REM
120 LINE(X1,Y1)-(X2,Y2)
130 IF DX+X1>255 THEN DX=-1*DX
140 IF DX+X1<0 THEN DX=-1*DX
150 IF EX+X2>255 THEN EX=-1*EX
160 IF EX+X2<0 THEN EX=-1*EX
170 IF DY+Y1>191 THEN DY=-1*DY
180 IF DY+Y1<0 THEN DY=-1*DY
190 IF EY+Y2>191 THEN EY=-1*EY
200 IF EY+Y2<0 THEN EY=-1*EY
210 X1=X1+DX
220 Y1=Y1+DY
230 X2=X2+EX
240 Y2=Y2+EY
250 REM
260 NEXT J
270 COLOR 15,4,7
280 END
```

これは画面上に線だけで絵を書くプログラムです。プログラムを実行すると、線が流れるように描かれていきます。

## フローチャート




変数リスト

Ｘ１，Ｙ１：線を引く、開始座標点
Ｘ２，Ｙ２：線を引く、終了座標点
DX，DY：開始点の移動幅
EX，EY：終了点の移動幅
J：ループカウント

考え方

線はＸ１、Ｙ１とＸ２、Ｙ２の間に引きます。線を引いたあと、次の線を引くために次の点を計算します。次の点が画面の範囲外になるときには、移動幅を逆にします（－１をかける)。

改造方法

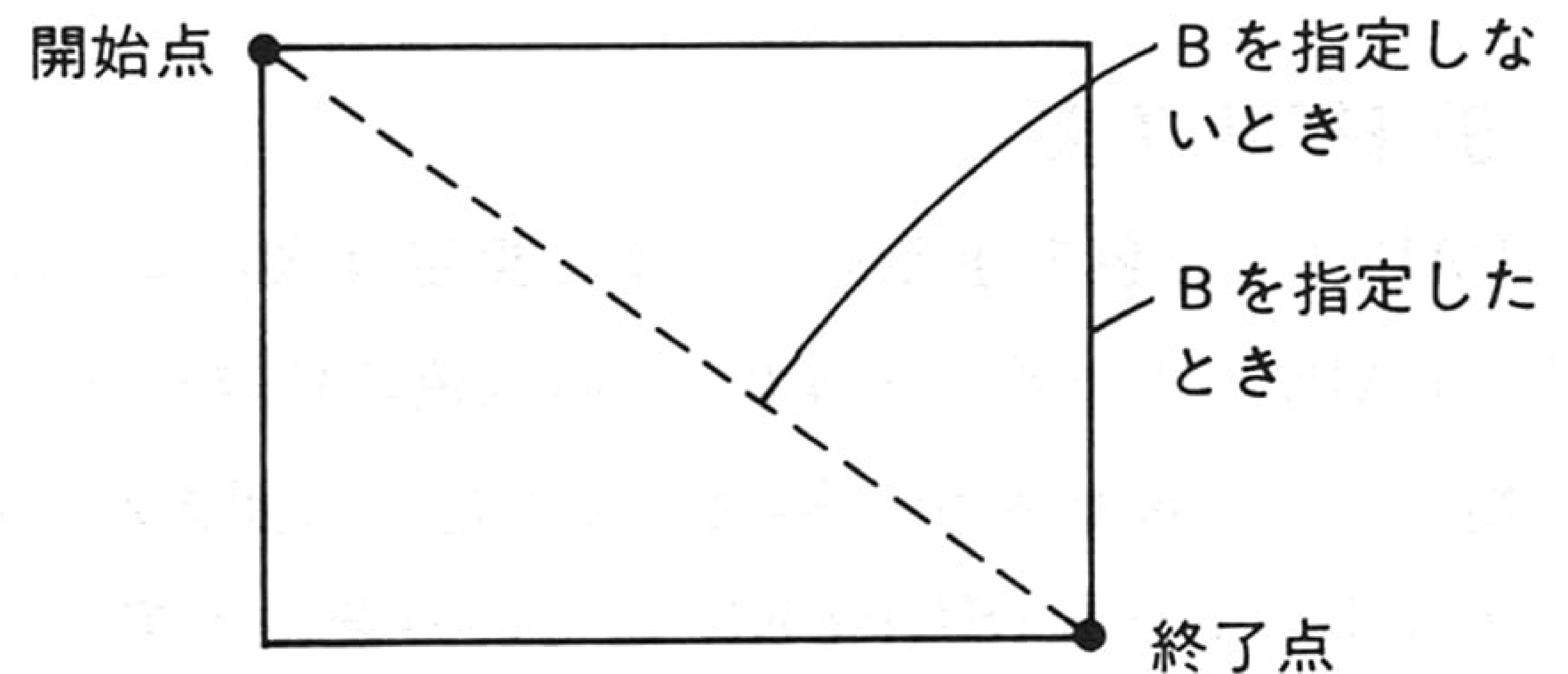
移動幅（DX、DY，EX、EY）を変更すれば、またちがうイメージの線画になります。

## (8) LINE②

LINE文は、単に線を引くだけでなく、四角形を描いたり、塗りつぶした四角形を描くこともできます。

四角形を描くとき　LINE(開始点)－(終了点)，色コード，B
塗りつぶした四角形　LINE(開始点)－(終了点)，色コード，BF

BはBOX(ボックス)を意味し、BFはBOXFILL(ボックスフィル)を意味します。開始点、終了点は、四角形の対角線の頂点を表わします。




LINE文を使って棒グラフを作ってみましょう。

```
10 REM ﾎﾞｳｸﾞﾗﾌ
20 DIM DA(5)
30 DA(1)=30
40 DA(2)=15
50 DA(3)=42
60 DA(4)=100
70 DA(5)=23
80 COLOR 15,1,1
90 SCREEN 2
100 FOR J=1 TO 5
110  LINE(20,J*20)-(20+DA(J)*(200/100),J*20+19),J+1,BF
120 NEXT J
130 GOTO 130
```

### プログラムの解説

20行　データを記憶するために配列を定義します。

30行～70行　配列にそれぞれのデータを記憶します。データは0～100の間で自由に変えることができます。

80行　画面の色を設定します。

90行　グラフィック画面にします。

100行　FOR～NEXTを使ってループのなかの処理を5回くり返します。

110行　LINE文で棒グラフを描きます。

開始X座標：20（20ドット目を基準にします。）

開始Y座標：J＊20（Jの値によって、20、40、60、80、100と変化します。）棒グラフの左上の座標です。

終了X座標：データの値が1に対して、画面上では2ドットにします。当然、座標は開始座標の値(20)を加える必要があります。

終了Y座標：棒グラフの幅を19ドットにしています。

色コード：色コードを2、3、4、5、6と変化させて棒グラフを作ります。

120行　FOR文に対応するNEXT文です。

130行　GOTO文でいつまでもプログラムを実行しています。グラフィック画面の状態を続けるためです。このプログラムを終了するときは CTRL → STOP を押します。

## (9) PAINT文

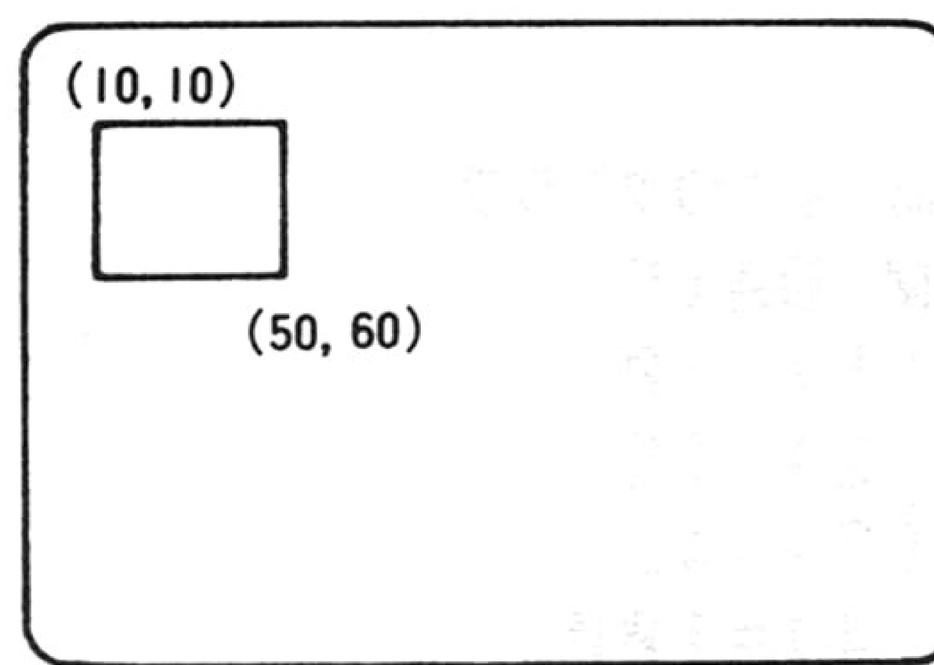
PAINT文は色を塗る命令です。指定は次のようにします。

PAINT(X座標，Y座標)，領域色の色コード，境界色の色コード

指定された座標を含む領域（境界色）に囲まれた領域を指定された領域色で塗ります。境界色はマルチカラーモード(SCREEN 3)でのみ使えます。

座標の指定は塗りつぶす範囲内であれば、どこの座標でも指定できます。

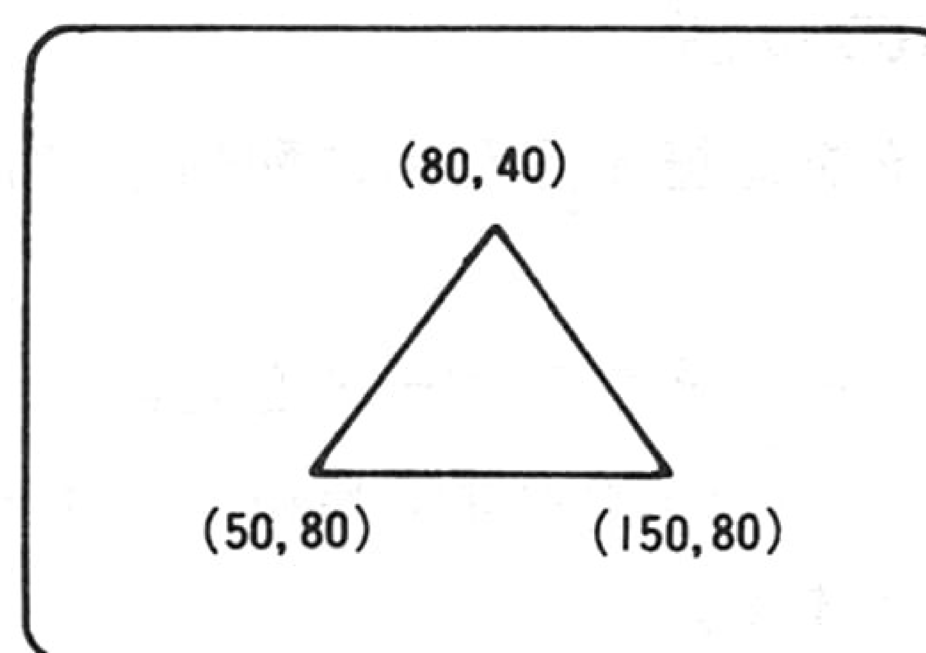
図の場合、四角のなかを塗りたいときの座標指定はX座標は11～49、Y座標は11～59の範囲であれば、どの値をとってもよいことになります。塗りつぶしたいときは、あらかじめ塗りつぶすわくを決めておかなければなりませんが、このわくがきちんと閉じていないと正しく塗りつぶされません。




```
10 COLOR 15,1,1
20 SCREEN 2
30 LINE(50,80)-(80,40)
40 LINE(80,40)-(150,80)
50 LINE(150,80)-(50,80)
60 PAINT(80,41),15
70 GOTO 70
```

このプログラムは、白い三角形のわくを書いたあとに白い色で塗りつぶすプログラムです。

このように、高解像度モード(SCREEN 2)ではわくの色と塗りつぶす色は同じでなければなりません。




## (10) CIRCLE文

いままでは、点を表示したり、直線を描いたりしましたが、今度は円です。CIRCLE文は円を描く命令です。指定は次のようになります。

CIRCLE (X座標，Y座標)，半径，色コード，開始角度，終了角度，比率

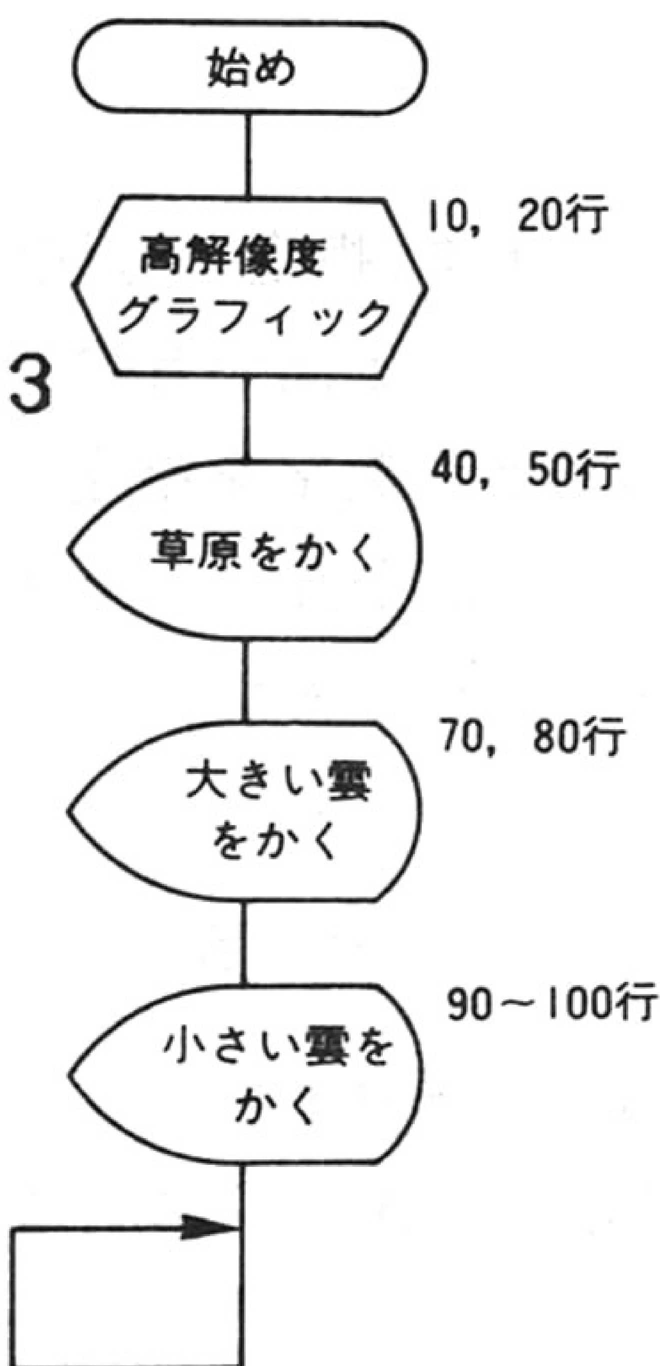
指定するデータがたくさんあるので、むずかしく感じるかもしれませんが、なれてしまえば簡単ですからしっかり覚えてください。

(X座標，Y座標)………円の中心点
半径………………………半径
色コード…………………円を描くときの色。指定を省略するとCOLOR文の前景色になります。
開始角度，終了角度……角度指定の単位はラジアンです。開始角度、終了角度は、円弧を描くときに指定します。この指定を省略すると円になります。
比率………………………比率は（垂直方向の半径）／(水平方向の半径）で指定します。比率が1以下のとき、半径は水平方向の半径を意味します。この場合、円は平たい円になります。また、比率が1以上のときは半径は垂直方向の半径になり、円はたて長になります。比率を省略すると円になります。

**プログラム**

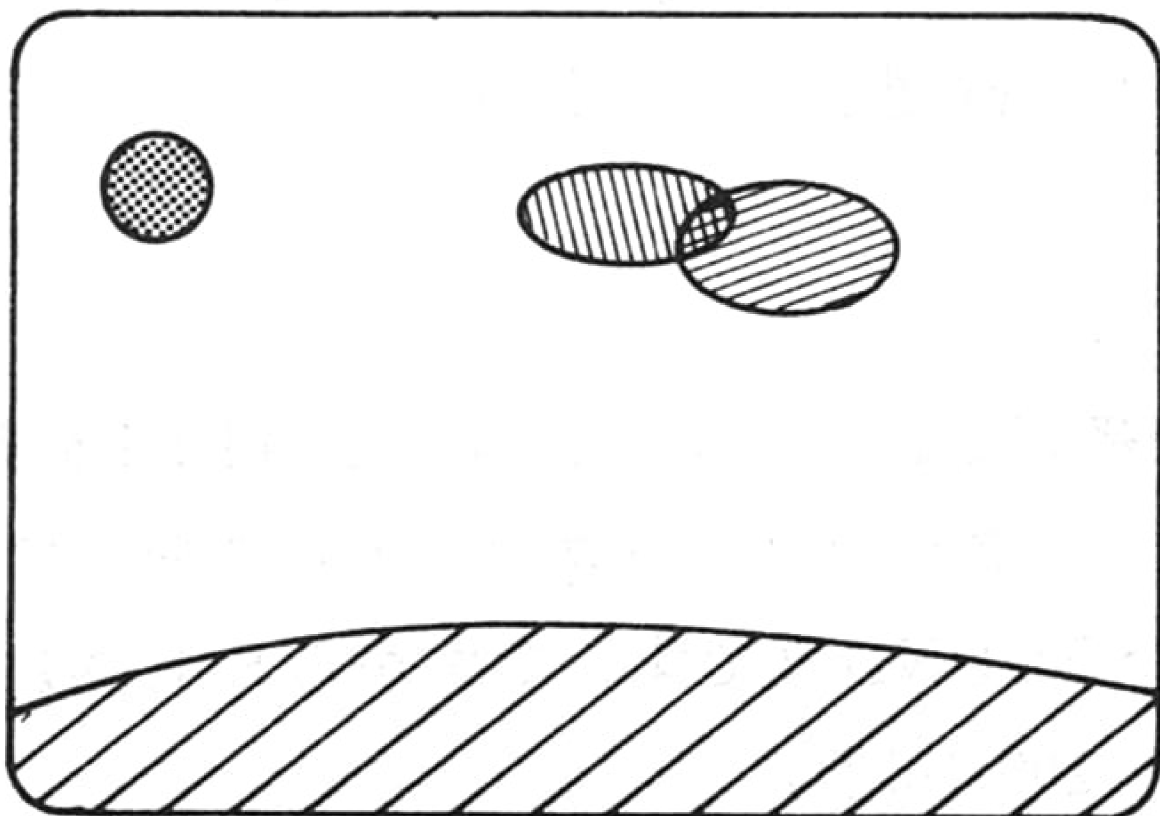
```
10 COLOR 15,7,7
20 SCREEN 2
30 REM ｿｳｹﾞﾝ
40 CIRCLE(256/2,191),128,3,,,1/3
50 PAINT(256/2,190),3
60 REM ｸﾓ
70 CIRCLE(170,30),40,15,,,1/3
80 PAINT(170,30),15
90 CIRCLE(200,50),30,15,,,1/3
100 PAINT(200,50),15
110 GOTO 110
```

**フローチャート**




**プログラム解説**

10、20行　高解像度グラフィックモードにします。色はうすい青にします。
40、50行　画面の下に緑で線を描き塗りつぶして草原のイメージにします。
70、80行　画面の上に白で線を描き、塗りつぶして雲のイメージを作ります。
90、100行　同じように雲を作ります。



草原も雲も、開始角度、終了角度を省略して円としていますが、比率は⅓にし、横長のだ円にしています。

**プログラムの改造**

画面の左上に太陽を描いてみてください。

**ワンポイント**

CIRCLE文で角度を指定するのに、ラジアンの単位を使いましたが、度で与えられた角度をラジアンで表わすにはどのようにしたらいいでしょうか。
180°は$\pi$ラジアンですから、1°は$\pi$/180ラジアンになります。
たとえば$x$°をラジアンで表わせば、$x$°=(3.1415926535898/180)* $x$ラジアンとなります。

## 5-3 音楽を楽しもう

MSXのもつ楽しい機能――音楽について説明しましょう。音に関しての命令はPLAY文とSOUND文があります。PLAY文は音楽を演奏する命令で、SOUND文は効果音を出すときに使う命令です。

### (1) PLAY文

PLAY文は、ドレミ…シに対応する音を記号で表わします。

```
ドレミファソラシ
| | | |  | | |
C D E F G A B
```

音の高さはオクターブで指定します。指定の方法はO(オクターブ)のあとに数字を指定します。数字は1～8の範囲です。数字が大きいほど高い音になります。半音あげるときは、A～Gのうしろに+か#をつけます。音の長さはLのあとに数字を指定します。このときの数字は1～64の範囲で、値が大きいほど音は短かくなります。L1が全音符、L2が2分音符、L4が4分音符となります。休符はR、そのあと数字を指定して長さを表わします。
PLAY文は次の書式で指定します。

PLAY 文字列1，文字列2，文字列3

ボイスチャンネルは1、2、3と3つありますから、文字列1から文字列3まで指定すると、同時に3重演奏ができます。文字列の指定は、A～Gの記号やL、R、Oなどを使います。(205P参照)

では、「ひのまる」の曲を演奏してみましょう。

**プログラム**

```
10 PLAY "O4L4CCDDEEDR4EEGGAAGR4"
20 PLAY "AAGGECDR4"
30 PLAY "GGECDEC"
40 END
```

文字列の指定にまちがいがあると、"Illegal function call" エラーとなります。そのときは文字列のなかを確認してください。

### (2) SOUND文

MX-101では3重和音、8オクターブの演奏ができるLSIが搭載されています。特にこのLSIをPSG(プログラマブルサウンドジェネレーター)と呼び、音をプログラマブル(コマンド)に発生できるLSIを意味しています。このPSGを自由にあやつる命令がSOUND文です。ですからSOUND文を操作するには、PSGの知識がどうしても必要になります。(209P参照)

SOUNDを使うと、さまざまな効果音を出すことができます。ゲームセンターで聞こえるあの音をMX-101で出すことができるのです。
書式は次の通りです。

SOUND　レジスタ番号，データ

レジスタ番号は、０～13で、データは0～255です。これは、指定したレジスタに、0～255の値を入れることを意味します。

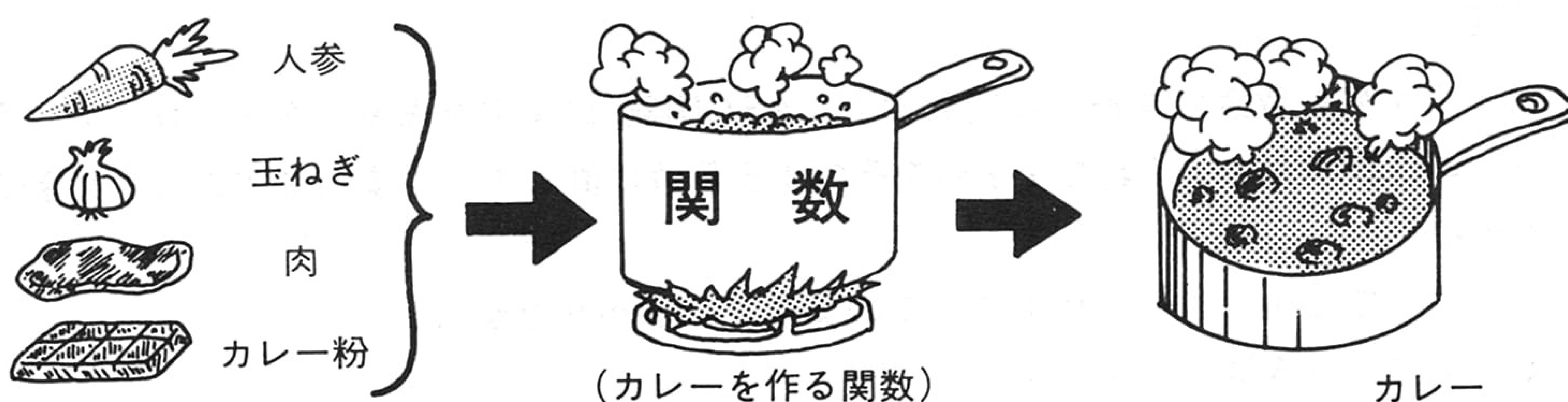
**プログラム**

```
10 REM UFO ノ オト
20 SOUND 0,65
30 SOUND 2,35
40 SOUND 4,68
50 SOUND 5,12
60 SOUND 6,19
70 SOUND 7,48
80 SOUND 8,10
90 SOUND 9,10
100 SOUND 10,16
110 SOUND 11,83
120 SOUND 12,23
130 SOUND 13,12
140 FOR J=1 TO 5000
150 NEXT J
160 SOUND 8,0
170 SOUND 9,0
180 SOUND 10,0
190 END
```

このプログラムは、UFOの音(?)を出すプログラムです。20行～130行で音の設定をしたあとに、140行～150行で時間待ちを行ない160行～180行で音を止めています。

## 5-4 関数

MSX BASICには、いろいろな関数が用意されています。これらのなかでも、よく使われる関数について説明します。
関数とは、ひとことでいえば「何かを与えることによって何かが返ってくる」という性質をもつものです。そして、この何かを引数（ひきすう）といいます。



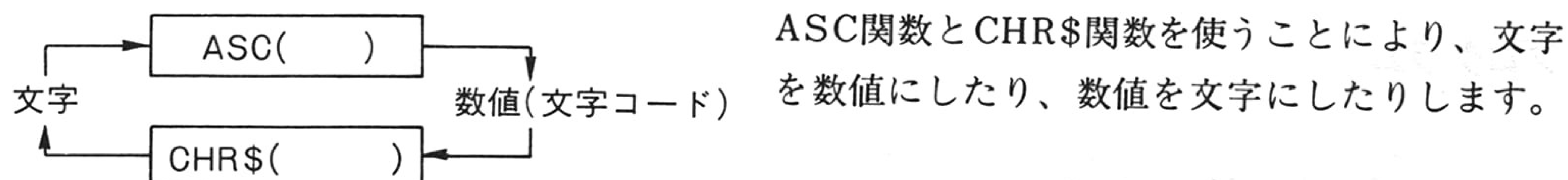
## (1) ASC関数

この関数は文字を与えて、その文字に対応する文字コード（数値）を返します。たとえば〝A〞は65になります。引数は先頭の1文字。

## (2) CHR$関数

ASC関数とは逆に、指定した数値に対応する文字を与えます。

ASC関数とCHR＄関数の関係




ASC関数とCHR$関数を使うことにより、文字を数値にしたり、数値を文字にしたりします。

### プログラム

```
10 FOR J=32 TO 255
20 IF J<100 THEN PRINT J;CHR$(J);"  ";
30 IF J>99 THEN PRINT J;CHR$(J);" ";
40 NEXT J
```

上のプログラムは、MX-101がもっている文字を文字コードと対応させ、画面に表示するプログラムです。

表示を中断するときは STOP キーを押してください。

文字コードは0～255までの範囲をとります。ただし、0～31までは特殊な意味をもちます（制御コード）。

### ワンポイント

グラフィック文字を表示するにはどのようにしたらよいでしょう。グラフィック文字コードは、0～31までの文字コードをもっていますが、ただ単にCHR＄（文字コード）を入力しても制御コードとみなされます。そのため、まずCHR$(1)を書き、そのあとに続けてグラフィックコードを書きます。そのグラフィックコードも64をたした値でなければなりません。たとえばπ(パイ)はグラフィックコードで16になります。ですから、PRINT CHR＄(1);CHR＄(16+64) になります。

```
10 FOR J=0 TO 31
20 PRINT J;CHR$(1);CHR$(J+64)
30 NEXT J
40 END
```

グラフィック文字をすべて表示するプログラムです。

## (3) VAL関数

VAL関数は、文字列を数値に変換する関数です。ASC関数は1文字をキャラクターコードに変換するものですが、VAL関数は与えられた文字列をそのまま数値に直します。また文字数も1文字でなくてもかまいません。

A＝ASC(〝1〞)ではAは49となります。(1の文字コードが49だから)

A＝VAL(〝1〞)ではAは1になります。

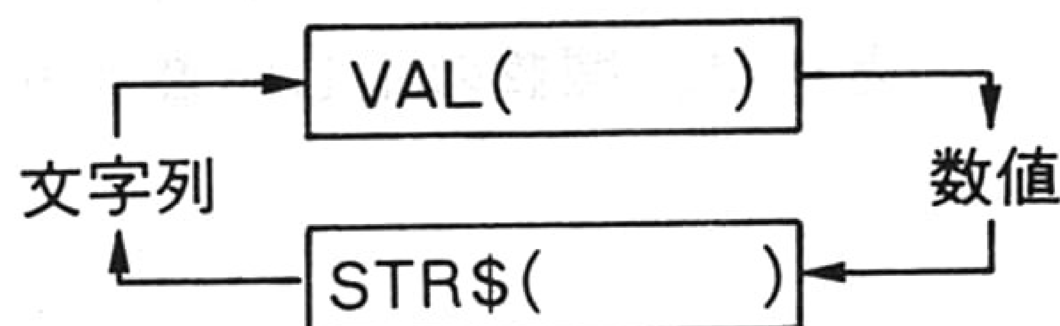
VAL関数の引数は文字列です。文字列のなかに数字以外の文字がでてくると、その文字のひとつ前までが有効です。

A＝VAL(〝123A56〞)のときはAの値が123になります。

## (4) STR$関数

VAL関数が文字列を数値に変換するのに対してSTR$関数は、数値を文字列にします。STR$関数の引数は数値です。数値が+のときは、STR$で作られた文字列の先頭は+を表わす空白になります。A$=STR$(123)を実行したときのA$の内容は"空白","1","2","3"の集まりになります。

VAL関数とSTR$関数の関係




### プログラム

```
10 INPUT "A=";A$
20 INPUT "B=";B$
30 A=VAL(A$)
40 B=VAL(B$)
50 C=A+B
60 D$=A$+B$
70 E$=STR$(A)+STR$(B)
80 PRINT "ｽｳﾁﾉ ﾀｼｻﾞﾝ(";C;")"
90 PRINT "ﾓｼﾞﾉ ﾀｼｻﾞﾝ(";D$;")"
100 PRINT "ﾓｼﾞﾉ ﾀｼｻﾞﾝ(";E$;")"
110 END
```

このプログラムは数値を文字という形で入力し、その文字列をVAL関数を使って一度数値におき換え、その数値のたし算をしています。また入力した2つの文字列を直接たした結果と、STR$関数を使って文字列に変換したあとにたした結果も表示します。

### プログラム解説

| | |
|---|---|
| 10～20行 | 数値を文字列として入力します。 |
| 30～40行 | 入力した文字列を数値に変換します。 |
| 50行 | 数値のたし算 |
| 60行 | 文字列のたし算 |
| 70行 | 数値を文字列に変換したあとのたし算 |
| 80～100行 | それぞれの表示 |
| 110行 | 終り |

### 実行例

```
A=? 123
B=? 12
ｽｳﾁﾉ ﾀｼｻﾞﾝ( 135 )
ﾓｼﾞﾉ ﾀｼｻﾞﾝ(12312)
ﾓｼﾞﾉ ﾀｼｻﾞﾝ( 123 12)
```

←123と、12の間があいているのは一度数値に直したため符号を表わす空白が入ったものです。

**ワンポイント**

〝Illegal function call〃と〝Type mismatch〃エラー

関数を使っていると、この2つのエラーがよく出ます。〝Illegal function call〃は、引数の値が範囲をこえたときに発生しますので、このときは引数の内容を確認してください。

一方〝Type mismatch〃は、引数の型が規則に反するときに発生します。引数が文字型でなければいけないのが、数値だったときなどです。このときは、関数の正しい書き方にもとづいてプログラムを修正してください。

```
10 A=-1
20 PRINT CHR$(A)
30 END
run
Illegal function call in 20
Ok
```

このプログラムは20行でエラーが発生しましたが、20行にまちがいがあるのではなく、Aの値が関数の範囲外だったためです。ですから、この場合Aの値を再検討する必要があります。

```
10 A=ASC(A)
20 PRINT A
30 END
run
Type mismatch in 10
Ok
```

このプログラムは10行目でエラーが発生しています。〝Type mismatch〃エラーですので、エラーがおきた行にまちがいがあります。ASC関数の引数は文字列でなければなりませんが、10行では数値になっています。

## (5) LEN関数

これは文字列の長さを表わす関数です。関数の指定は次のようになります。

A＝LEN（文字列）

たとえば、A＝LEN（〝ABC〃）とすれば、Aの値は3になります。

## (6) LEFT$関数

LEFT$は、指定した文字列のなかの左側から指定した分だけ抜きだします。

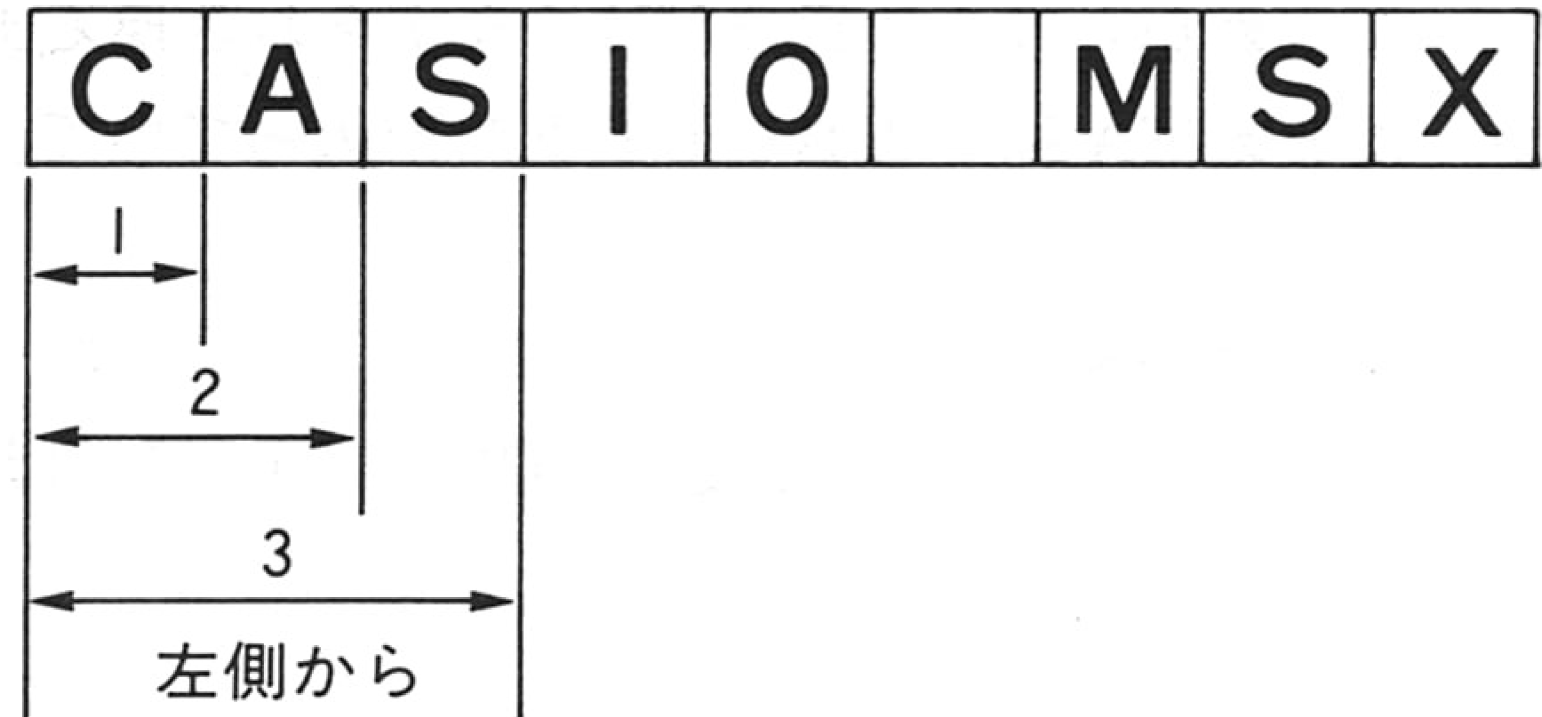
書き方は次の通り。

LEFT$（文字列，文字数）

**プログラム**

```
10 A$="CASIO MSX"
20 N=LEN(A$)
30 FOR J=1 TO N
40    B$=LEFT$(A$,J)
50    PRINT J;"(";B$;")"
60 NEXT J
70 END
run
 1 (C)
 2 (CA)
 3 (CAS)
 4 (CASI)
 5 (CASIO)
 6 (CASIO )
 7 (CASIO M)
 8 (CASIO MS)
 9 (CASIO MSX)
Ok
```

このプログラムはA$に〝CASIO MSX〞を記憶させ、その内容を左から1文字、2文字と表示するプログラムです。LEFT$関数の働きがよくわかると思います。




## (7)RIGHT$関数

RIGHT$関数は、指定した文字列のなかの右側から指定した文字分だけ抜き出します。書き方は次の通りです。

RIGHT$（文字列，文字数）

## (8)MID$関数

MID$関数は、文字列のなかから指定した分だけとり出します。

**プログラム**

```
10 A$="CASIO MSX"
20 N=LEN(A$)
30 FOR J=1 TO N-1
40    B$=MID$(A$,J,2)
50    PRINT J;"(";B$;")"
60 NEXT J
70 END
run
 1 (CA)
 2 (AS)
 3 (SI)
 4 (IO)
 5 (O )
 6 ( M)
 7 (MS)
 8 (SX)
Ok
```

このプログラムはMID$関数を使って、文字列のなかから2文字を表示するものです。

MID$の書式は次の通り。

MID$（文字列，何文字めから，何文字分）――①

または、

MID$(文字列，何文字めから全部)――――②

①は文字列のなかの特定文字をとり出す指定です。

②は文字列の何文字めから全部を取り出す指定方法です。

**ワンポイント**

MID$は関数以外に、ステートメントとしても用意されます。このときの指定方法は、次のようになります。

　MID$（文字列，式1，式2）＝文字列

この命令は文字列の一部を他の文字列でおき換えるときに使います。文字列のなかの式1文字めから式2の値分だけの文字をおき換えます。式2は省略できます。

たとえばA$＝〝ABCDEF〟が入っていて、このなかの〝CD〟を〝XY〟とおき換えるときは、

　MID$（A$，3，2）＝〝XY〟とします。

A$の3文字目から2文字分CDを〝XY〟におき換えます。

## (9) HEX$関数

数値を表現するのに私たちがいちばんよく使う10進数(0～9)のほかにMSXでは2進数、8進数、16進数を使うことができます。(155P参照)

HEX$関数は数値を16進表記の文字列に変換する関数です。書式は次の通り。

　HEX$（数値）

**プログラム**

```
10 INPUT"10シンスウ";A
20 IF A=0 THEN 60
30 H$=HEX$(A)
40 PRINT"16シンスウ   ";H$
50 GOTO 10
60 END
```

このプログラムを実行すると10進数を入力するようにメッセージが表示されますので、10進数を入力してください。その10進数に対応する16進数が表示されます。

0を入力すると、プログラムの実行を終了します。

## (10) OCT$関数

この関数は数値を8進表記の文字列に変換する関数です。書式は次の通り。

　OCT$（数値）

## (11) BIN$関数

この関数は数値を2進表記の文字列に変換する関数です。書式は次の通り。

　BIN$（数値）

## まとめ

2進数、16進数、8進数および10進数はすべて数値ですが、BIN$、HEX$、OCT$関数で与えられたデータは文字列になります。数値として2進数、16進数、8進数を表わすときは、&B、&H、&Oを数字の先頭につけます。

例)

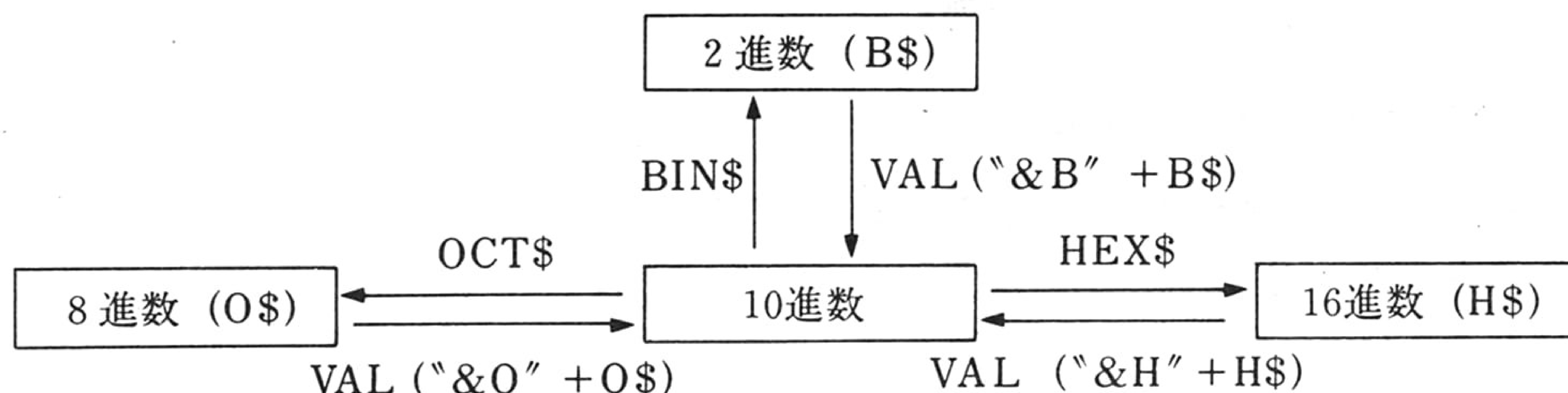
| 2進数 | 8進数 | 16進数 | 10進数 |
|---|---|---|---|
| &B1010 | &O12 | &HA | 10 |
| &B1001 | &O11 | &H9 | 9 |

10進数は、0、1、2、3、4、5、6、7、8、9から構成されます。

2進数は、0、1から構成されます。

8進数は、0、1、2、3、4、5、6、7で構成されます。

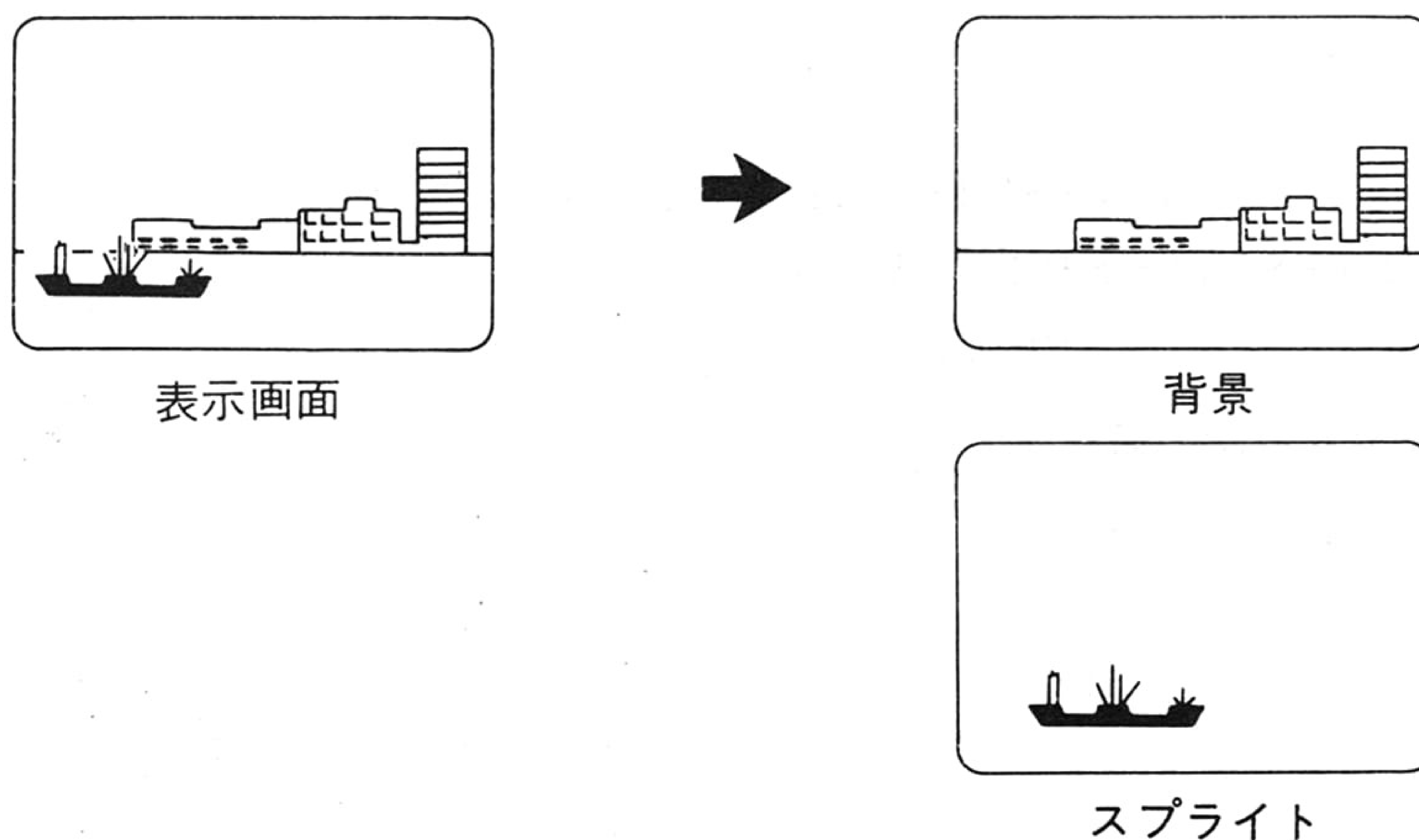
16進数は、0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、Fから構成されます。




# 5-5 スプライト

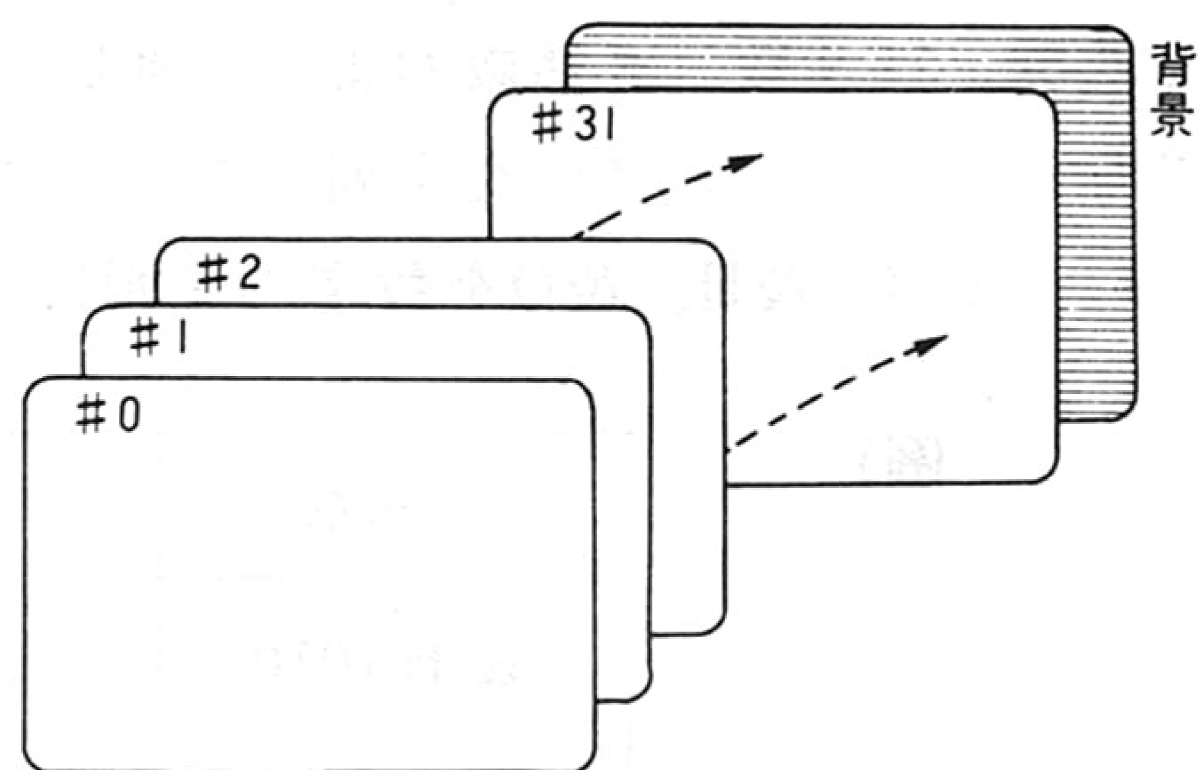
## (1) スプライト機能

MSX BASICの最大の特長ともいえるスプライトについて説明します。スプライトは「動画」とも呼ばれるように、画面上のスプライトキャラクターを自由に動かすことができるものです。
LINE文やPSET文、CIRCLE文などで描いた図形は、いずれも背景といって動きのないものでしたが、スプライトは背景の上を自由に動かすことができるのです。



スプライトは最大32個まで表示することができますが、それぞれのスプライト表示画面には優先順位があります。

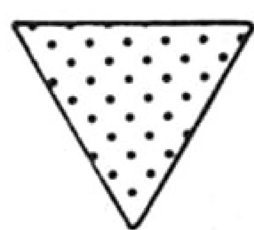
つまり、＃０～背景まで画面が33枚重なっていると考えるとよいでしょう。ですから、スプライトが２つ重なったときは、優先度の高い方が表示されます。その優先度はスプライト番号０がいちばん高く、あとは番号を追うごとに低くなり、31がいちばん低くなります。

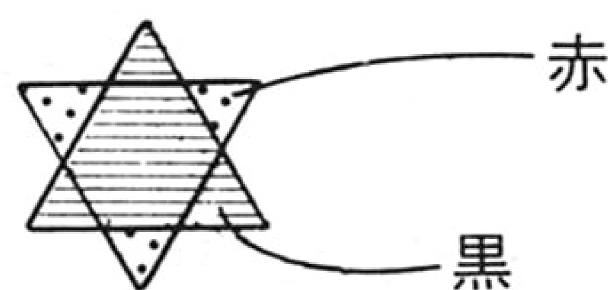


〔＃0～＃31をスプライト表示面と呼びます〕

たとえば、スプライト画面＃０に▲の形を黒で表示し、＃１に▼の形を赤で表示すると、重なったところは優先度の高い黒となります。


（黒）


（赤）




## (2) スプライトキャラクターの大きさ

スプライト機能で自由に動かすことができるキャラクターのパターンの大きさは、４種類のなかから選ぶことができます。

●パターンサイズ

0　８×８ドット標準
1　８×８ドット拡大(16×16ドットの大きさ)
2　16×16ドット標準
3　16×16ドット拡大(32×32ドットの大きさ)

このスプライトパターンサイズはSCREEN文で定義します。

SCREEN　*n*，パターンサイズ　——*n* は画面モード

スプライトは、８×８ドットで定義するときは、最大256種類のパターンが定義できます。ただし、16×16ドットの大きさのときは最大64種類になります。

## (3) スプライトの定義

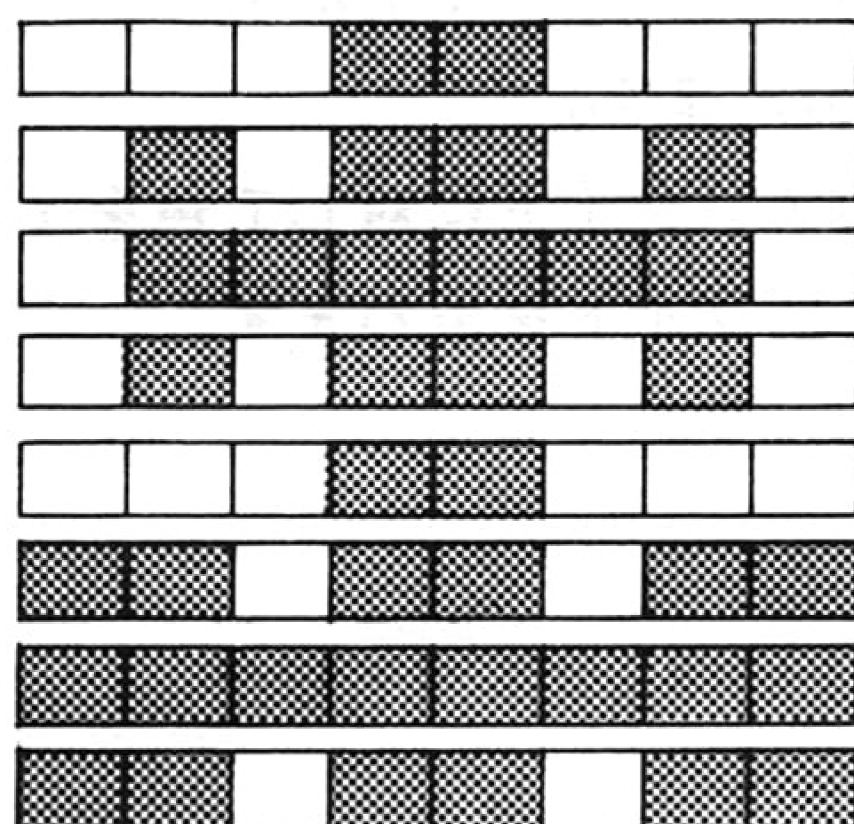
スプライトの定義は予約変数のSPRITE$に値を入れる方法で定義します。

８×８ドット——SPRITE$(０)～SPRITE$(255)
16×16ドット——SPRITE$(０)～SPRITE$(63)

スプライト変数を定義する式は次の通り。

SPRITE$(n)＝文字例

nは0～255または0～63の範囲です。(優先順ではありません)
文字列は、８×８ドットのときは８文字、16×16ドットのときは32文字になります。



●8×8ドットの定義方法

8×8ドットのパターンを横に8ドットごとにわけ、点があるところを〝1〟とし、点のないところを〝0〟とします。そしてその〝1〟と〝0〟を並べて2進数にします。この2進数を、CHR$関数で文字にします。

たとえば左のパターンをスプライト(0)に定義するには次のようになります。

| 2進表記 | 16進表記 | 10進表記 |
|---|---|---|
| &B00011000 → | &H18 → | CHR$(24) |
| &B01011010 → | &H5A → | CHR$(90) |
| &B01111110 → | &H7E → | CHR$(126) |
| &B01011010 → | &H5A → | CHR$(90) |
| &B00011000 → | &H18 → | CHR$(24) |
| &B11011011 → | &HDB → | CHR$(219) |
| &B11111111 → | &HFF → | CHR$(255) |
| &B11011011 → | &HDB → | CHR$(219) |

SPRITE$(0)=CHR$(&B00011000)+CHR$(&B01011010)+CHR$(&B01111110)+……CHR$(&B01011010)となります。

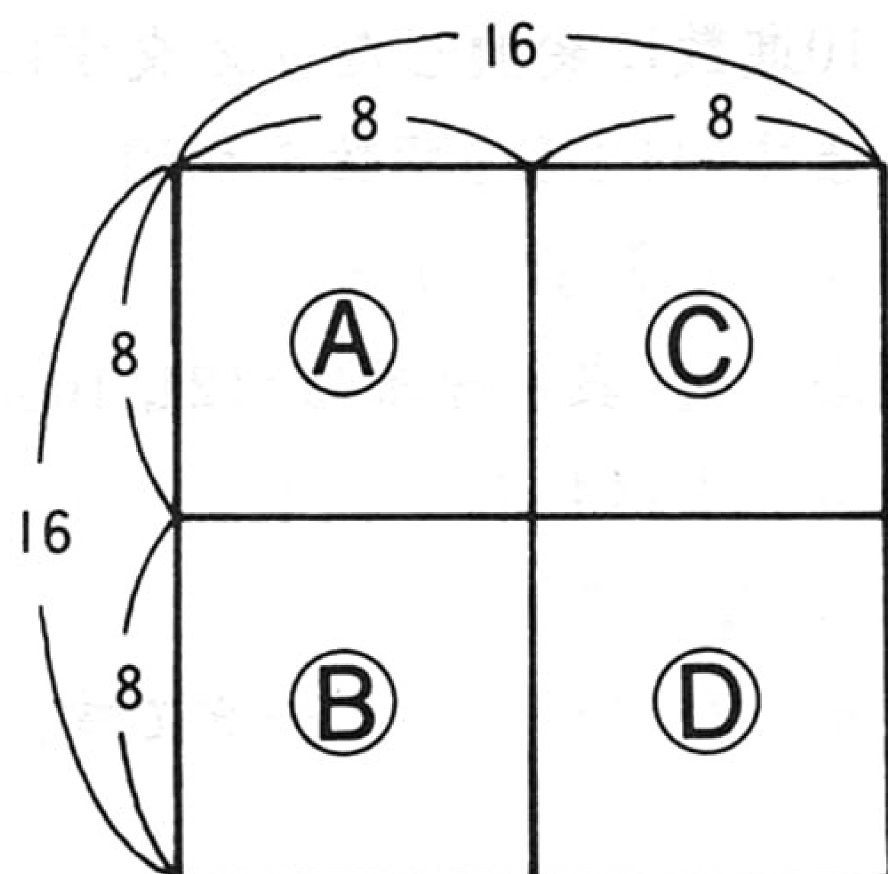
2進数で表現すると式が長くなるので、8進数や16進数で表現してもよいでしょう。

10進数のときは次のようになります。

SPRITE$(0)=CHR$(24)+CHR$(90)+CHR$(126)+…CHR$(219)

●16×16ドットの定義方法

16×16ドットのときは、全体を8×8ドットごとに分割すると考えます。




Ⓐ、Ⓑ、Ⓒ、Ⓓをそれぞれ8×8ドットで定義し、これをつなげてスプライト変数に代入します。

たとえば、スプライトの予約変数5に定義するときは

SPRITE$(5)= Ⓐの文字列 + Ⓑの文字列 + Ⓒの文字列 + Ⓓの文字列

8文字　8文字　8文字　8文字

合計32文字が必要です。

## (4) スプライト画面の表示方法

スプライトを画面に表示するには、PUT SPRITE文を使います。書式は次の通り。

PUT SPRITE スプライト表示面，(X座標，Y座標)，色コード，パターン番号

**スプライト表示面**…スプライトを表示する面の番号で0～31の範囲です。番号が小さいほど表示の優先度が高くなります。ですから2つのスプライトが重なったときは手前に表示された方が優先されます。

**スプライトの表示座標の指定**…X座標とY座標の指定で行ないます。ただし、画面の左上の座標は(0，1)になります。これはグラフィックの座標とは上下に1ドットずれていますので気をつけてください。

Y座標に208を指定すると、指定した表示画面以降（優先順が低い画面）のスプライトが消えます。Y座標に209を指定すると、指定したスプライト表示面のスプライトが消えます。

**色コード**…スプライトは、ひとつのスプライト表示面にあるひとつのスプライトに対して色を1色だけつけることができます。したがって、2色つけたいときは2つのスプライトを重ねます。
省略すると、前に指定した色になります。

**パターン番号**…スプライトを定義したパターンの番号です。

### (5)スプライト表示の順序

1.スプライトパターンのサイズを決める———SCREEN文を使う
2.スプライトパターンの定義———————SPRITE$の値を設定
3.スプライトパターンの表示———————PUT SPRITE文を使う

では(3)スプライトの定義で使った車のパターンをもとに8×8ドットのスプライトを表示してみます。

**プログラム**

```
10 COLOR 15,1,1
20 SCREEN 1,0
30 FOR J=1 TO 8
40 READ A$
50 N$=N$+CHR$(VAL("&B"+A$))
60 NEXT J
70 SPRITE$(0)=N$
80 PUT SPRITE 0,(121,100),4,0
90 DATA 00011000
100 DATA 01011010
110 DATA 01111110
120 DATA 01011010
130 DATA 00011000
140 DATA 11011011
150 DATA 11111111
160 DATA 11011011
```

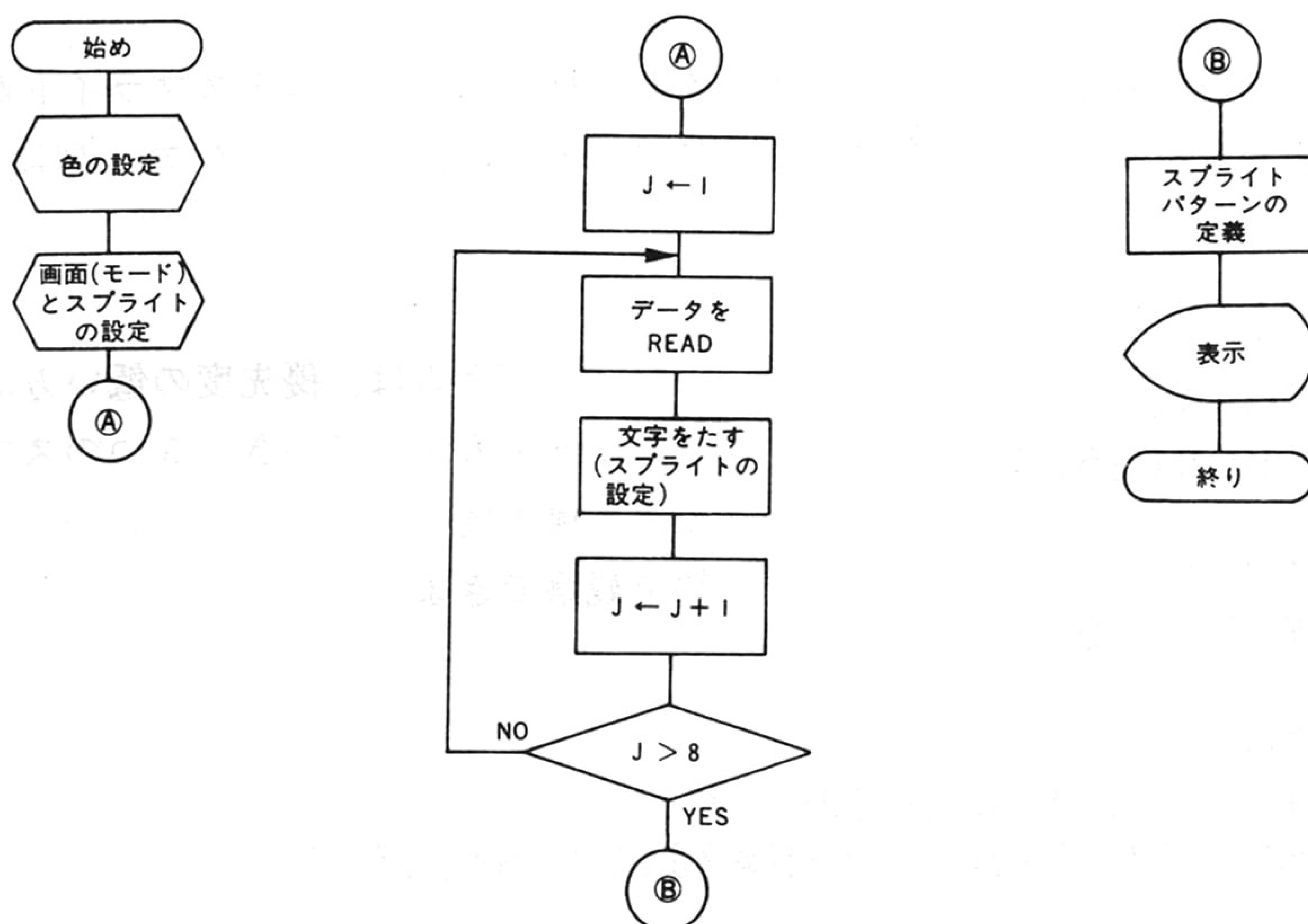
10～20行　画面を設定します。SCREEN 1, 0はテキスト画面に、8×8ドットのスプライトを表示する指定です。

30～70行　スプライトパターンをスプライト変数の0に定義します。データは2進数で入っているため一度〝&B″をつけて2進文字列とし、VAL関数を利用して10進数に変換したうえ文字にしています。(CHR$関数を利用)

80行　スプライトを表示します。
表示画面は0、表示座標は(121, 100)、表示色コードは4、スプライトパターンは0、になります。

90～160行　スプライトパターンのデータです。

**フローチャート**

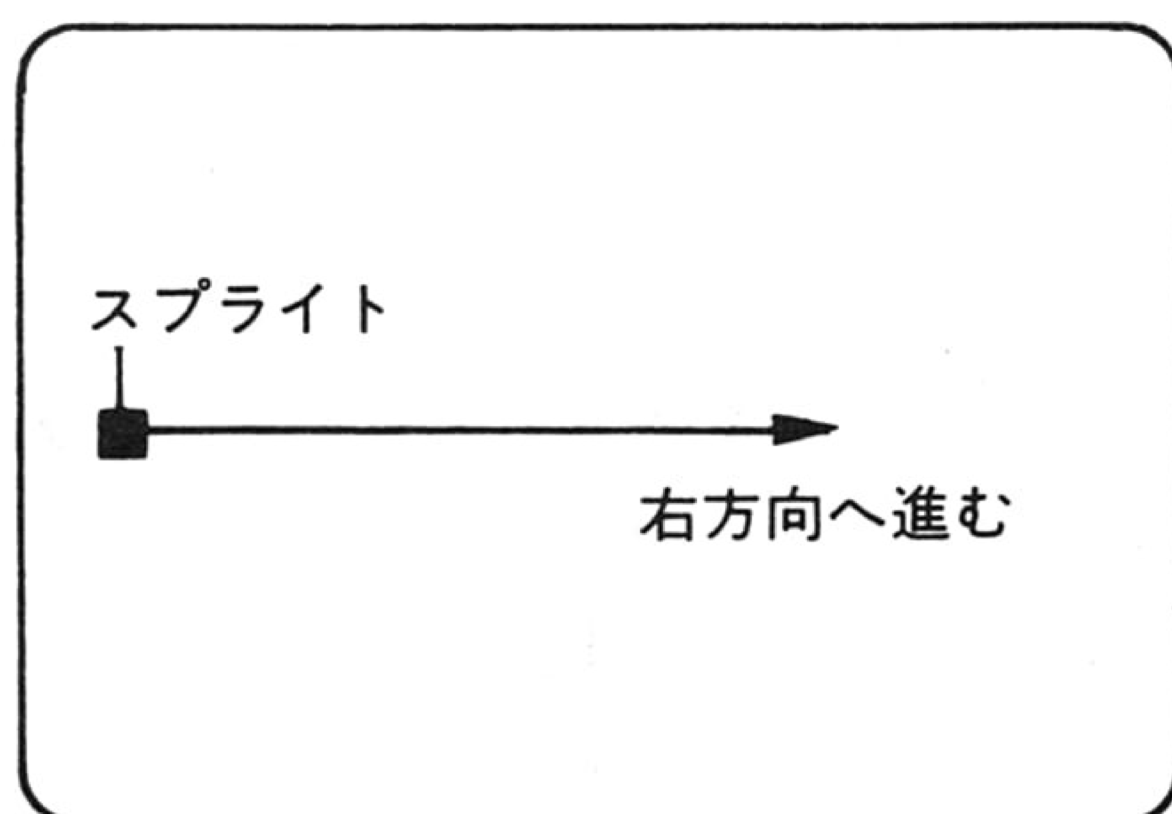



## (6) スプライトを動かす

スプライトを動かすにはPUT SPRITE文の座標を変更するだけで簡単にできます。
これを連続して行なうには座標を連続して変化させればいいのです。

**プログラム**

```
10 COLOR 15,1,1
20 SCREEN 1,1
30 FOR J=1 TO 8
40   A$=A$+CHR$(255)
50 NEXT J
60   SPRITE$(0)=A$
70 FOR X=1 TO 255
80   PUT SPRITE 0,(X,50),8,0
90 NEXT X
100 COLOR 15,4,7
110 END
```



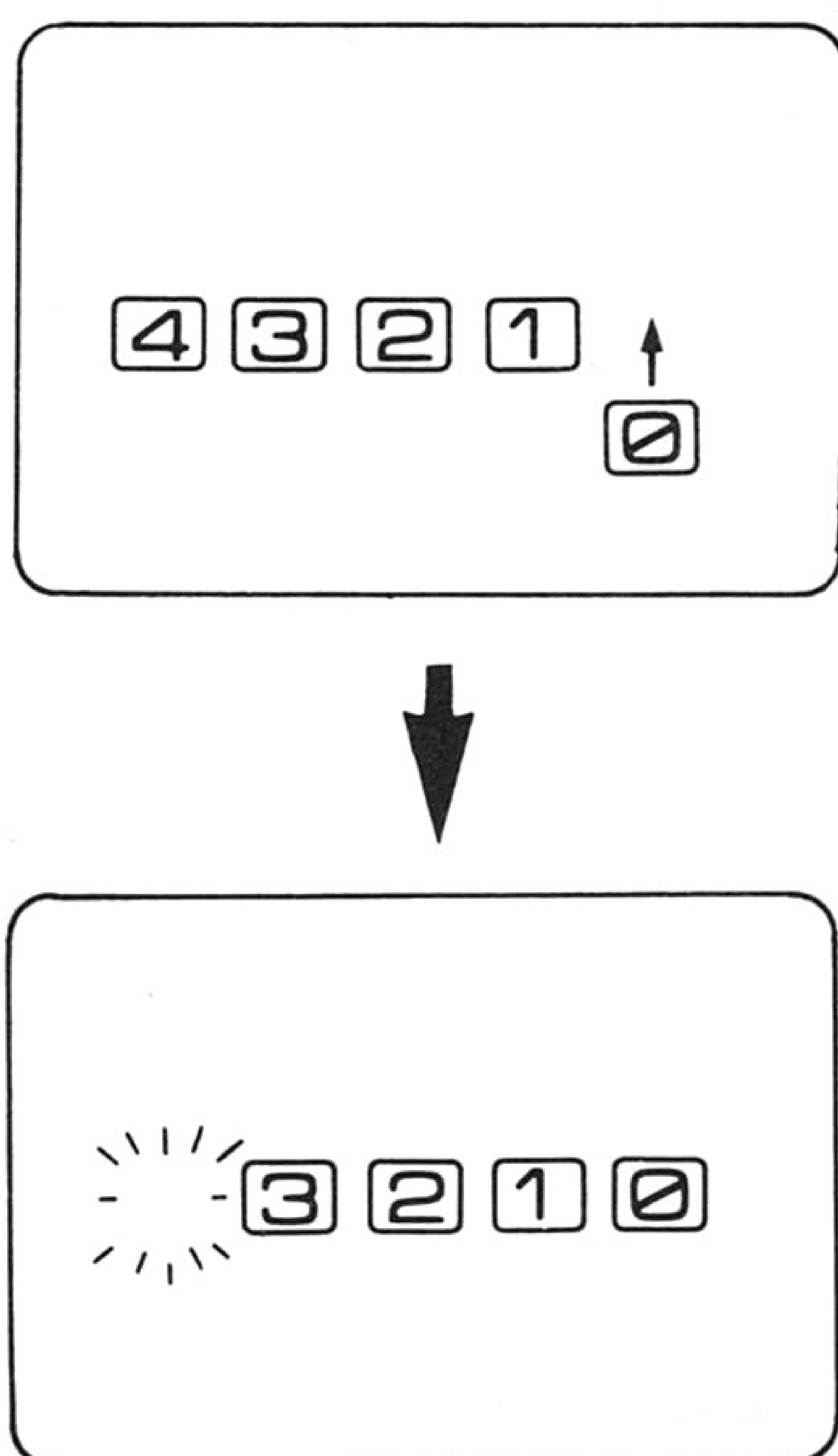

| | |
|---|---|
| 10～20行 | 画面とスプライトサイズの設定。 |
| 30～60行 | スプライト変数の"0"にスプライトパターンを定義します。 |
| 70行 | FOR文でX座標を1から255まで変化させています。 |
| 80行 | スプライトの表示をするのに、X座標の値を変数としています。 |
| 90行 | FOR文に対応するNEXT文です。 |
| 100行 | 色を変更します。 |
| 110行 | プログラムを終了します。 |

### (7) スプライトの制約

スプライトは、32個まで同時に表示することができますが、横一列に5つ以上スプライトが並ぶと、優先度の低いものから消えてしまいます。なぜなら、スプライトは最高4つしか 横一列に並ぶことができないのです。

```
10 COLOR 15,1,1
20 SCREEN 2,1
30 FOR J=1 TO 8
40   A$=A$+CHR$(255)
50 NEXT J
60 FOR S=0 TO 4
70   SPRITE$(S)=A$
80 NEXT S
90 FOR S=1 TO 5
100   FOR Y=191 TO 100 STEP -1
110     PUT SPRITE 5-S,(20+S*20,Y),S+3,5-S
120   NEXT Y
130 NEXT S
140 END
```

このプログラムは、優先度の低い方から順次スプライトを表示していき、5つのスプライトが並ぶと優先度の低いスプライトが消えてゆく状態を観察できます。

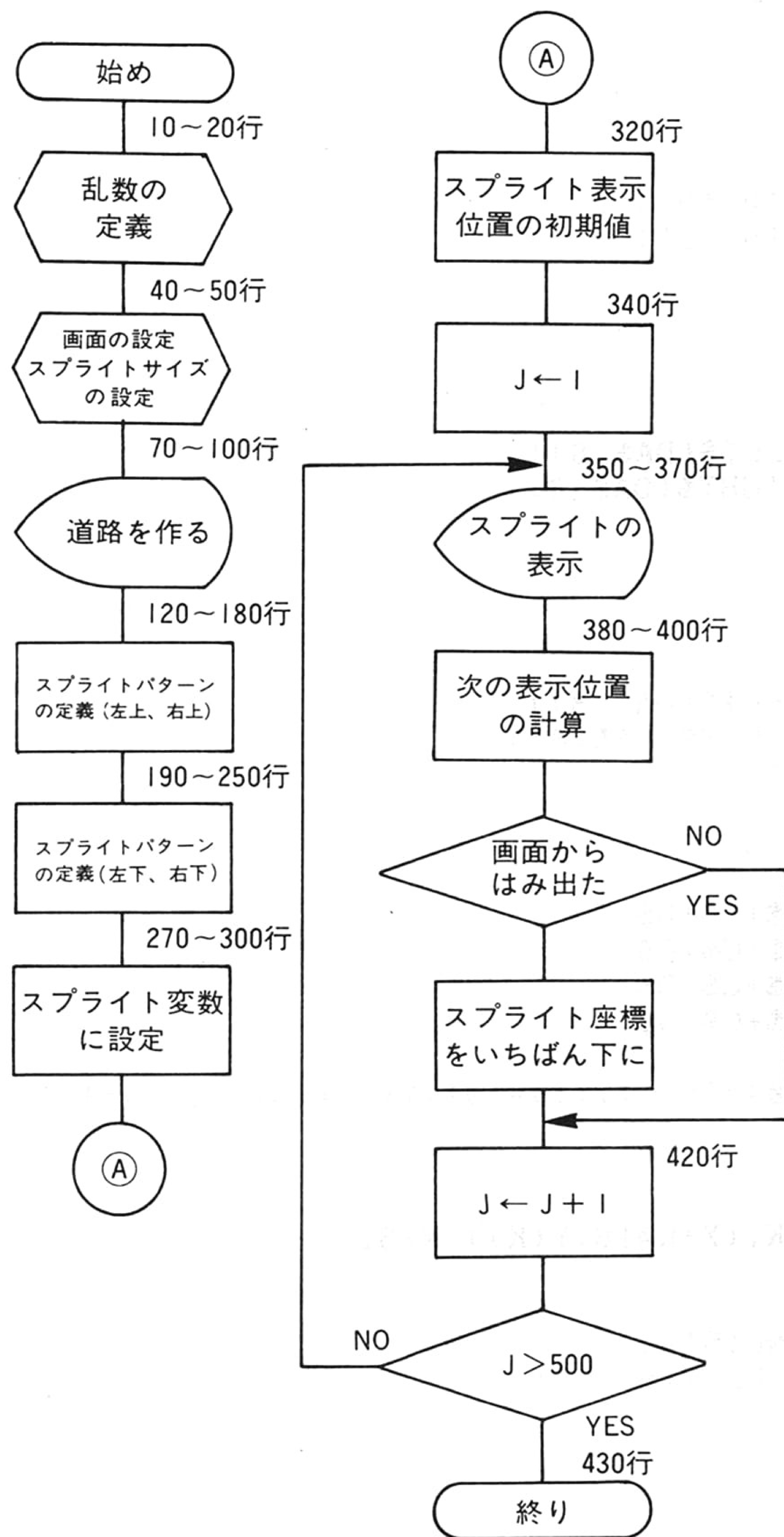



4、3、2、1、0 は優先順を表わし、0が最も高いスプライトです。

### (8) サンプルプログラム

16×16ドットのスプライトを定義し、このスプライトを動かすプログラムを作ってみました。このプログラムは単純ですので、どなたでも理解しやすいと思います。
そして、このプログラムが理解できれば、スプライトに関しては、もう卒業です。

```
10 DEF FNR(X)=INT(RND(1)*X)
20 R=RND(-TIME)
30 REM
40 COLOR 15,1,1
50 SCREEN 2,2
60 REM
70 LINE(8*10,0)-(8*10,191),2
80 LINE(8*19,0)-(8*19,191),2
90 PAINT(0,0),2
100 PAINT(255,0),2
110 REM
120 FOR J=1 TO8
130   READ DA$
140   A1=VAL("&B"+LEFT$(DA$,8))
150   C1=VAL("&B"+RIGHT$(DA$,8))
160 A$=A$+CHR$(A1)
170 C$=C$+CHR$(C1)
180 NEXT J
190 FOR J=1 TO 8
200   READ DA$
210   B1=VAL("&B"+LEFT$(DA$,8))
220   D1=VAL("&B"+RIGHT$(DA$,8))
230 B$=B$+CHR$(B1)
240 D$=D$+CHR$(D1)
250 NEXT J
260 REM
270 SPRITE$(0)=A$+B$+C$+D$
280 SPRITE$(1)=A$+B$+C$+D$
290 SPRITE$(2)=A$+B$+C$+D$
300 SPRITE$(3)=A$+B$+C$+D$
310 REM
320   X=10*8+4 : Y(0)=191 :Y(1)=191:Y(2)=191 :Y(3)=191
330 REM
340 FOR J=1 TO 500
350   FOR K=0 TO 3
360     PUT SPRITE K,(X+K*16,Y(K)),K+3,K
370   NEXT K
380   FOR K=0 TO 3
390     Y(K)=Y(K)-FNR(5)*3
400     IF Y(K)<0 THEN Y(K)=191
410   NEXT K
420 NEXT J
430 END
440 REM
450 DATA 0000000110000000
460 DATA 0000001111000000
470 DATA 0011001111001100
480 DATA 0011111111111100
490 DATA 0011111111111100
500 DATA 0011001111001100
510 DATA 0000001111000000
520 DATA 0000001111000000
530 DATA 0111001111001110
540 DATA 0111001111001110
550 DATA 0111111111111110
560 DATA 0111111111111110
570 DATA 0111001111001110
580 DATA 0111001111001110
590 DATA 0000111111110000
600 DATA 0000111111110000
```


始め
10～20行
乱数の
定義
40～50行
画面の設定
スプライトサイズ
の設定
70～100行
道路を作る
120～180行
スプライトパターン
の定義（左上、右上）
190～250行
スプライトパターン
の定義（左下、右下）
270～300行
スプライト変数
に設定
Ⓐ
Ⓐ
320行
スプライト表示
位置の初期値
340行
J←1
350～370行
スプライトの
表示
380～400行
次の表示位置
の計算
画面から
はみ出た
NO
YES
スプライト座標
をいちばん下に
420行
J←J＋1
NO
J＞500
YES
430行
終り

## プログラムの解説

このプログラムは16×16ドットのＦ１レースカーを４台走らせるプログラムです。レーシングカーは画面の下から上へ走っていきます。いちばん上までくると再び下から現われます。

10行　乱数の関数を定義します。

20行　新しい系列の乱数を発生させます。（乱数で発生する数値をそのつど変化させる）

40行　色を設定

50行　画面を高解像度グラフィックモードにし、スプライトサイズを16×16ドットに設定します。

70〜100行　道路を作っています。

120〜180行　スプライトの左上、右上のパターンを定義します。

190〜250行　スプライトの左下、右下のパターンを定義します。

| | |
|---|---|
| Ⓐ | Ⓒ |
| Ⓑ | Ⓓ |

Ⓐ、Ⓒは120〜180行で定義

Ⓑ、Ⓓは190〜250行で定義

270〜300行　スプライト変数の０〜３にレーシングカーを定義しています。

320行　スプライトを表示するための最初の座標を定義します。

340行　処理を500回くり返します。

350〜370行　スプライトを表示します。スプライトの表示画面番号と表示パターンは対応しています。

スプライト変数０→表示画面０
スプライト変数１→表示画面１
スプライト変数２→表示画面２
スプライト変数３→表示画面３

色コードは、3、4、5、6を使っています。

380〜400行　レーシングカーの進む距離を乱数で決定しています。乱数の発生は、０〜４までになります。この乱数に３をかけていますので、レーシングカーの進む距離は、0、3、6、9、12のいずれかになります。画面からはみだすと、再び画面の下にもどします。

420行　NEXT文です。

430行　プログラムの終り

450〜600行　レーシングカーの定義

### 変数リスト

R：特に意味を持ちません。乱数系列を新しくします。

DA$：レーシングカーのパターンをREAD文で読むときに使う変数です。

A１：パターンのデータ（左上）を0〜255の数値に変換します。

B１：パターンのデータ（左下）を0〜255の数値に変換します。

C１：パターンのデータ（右上）を0〜255の数値に変換します。

D１：パターンのデータ（右下）を0〜255の数値に変換します。

A$：パターンの左上のデータ

B$：パターンの左下のデータ

C$：パターンの右上のデータ

D$：パターンの右下のデータ

X：スプライトを表示するためのX座標の基準になる値（道路の左側）

Y(0)：１台めのレーシングカーのY座標

Y(1)：２台め　〃

Y(2)：３台め　〃

Y(3)：４台め　〃

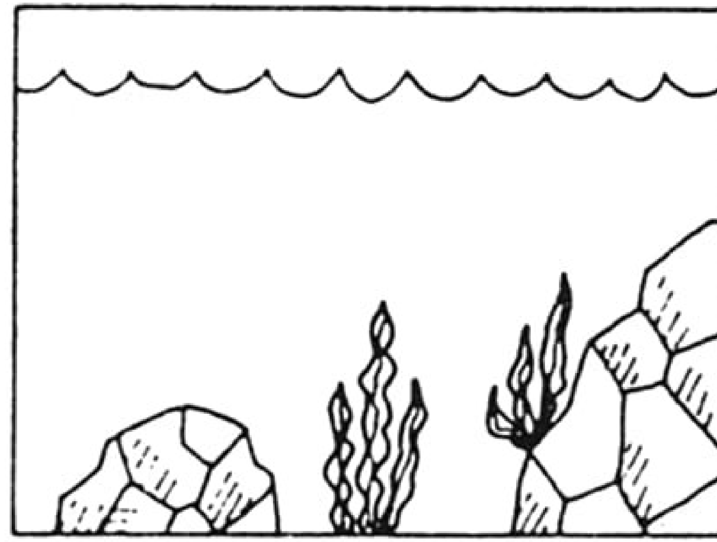
I：ループ変数

K：ループ変数、処理するスプライトの番号に対応しています。

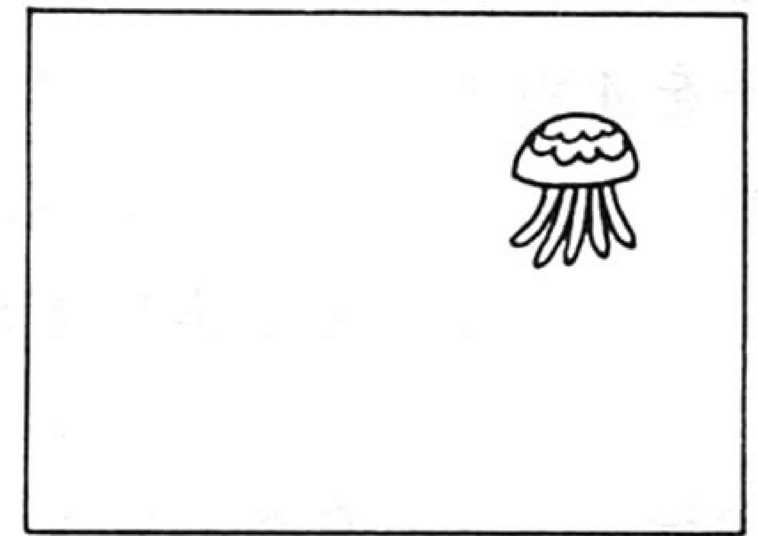
## スプライトのまとめ

海の中の動きを例にスプライトについてまとめてみましょう。

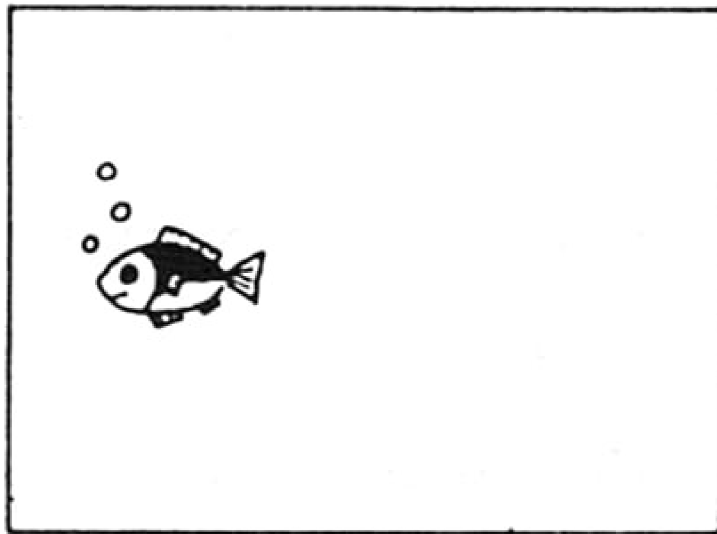
①画用紙の上に海の中の絵を描きます。この絵を〝背景〟と呼びます。



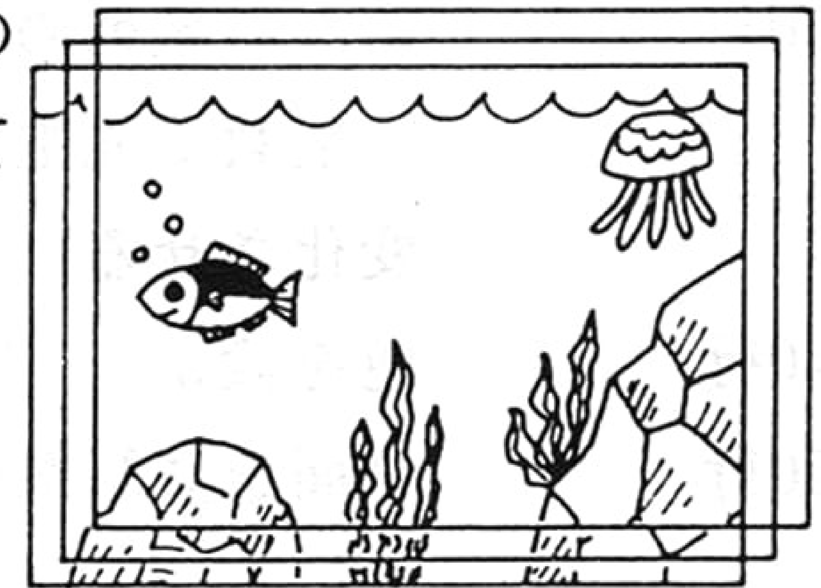
③同じように透明なセルロイドにクラゲを描きます。



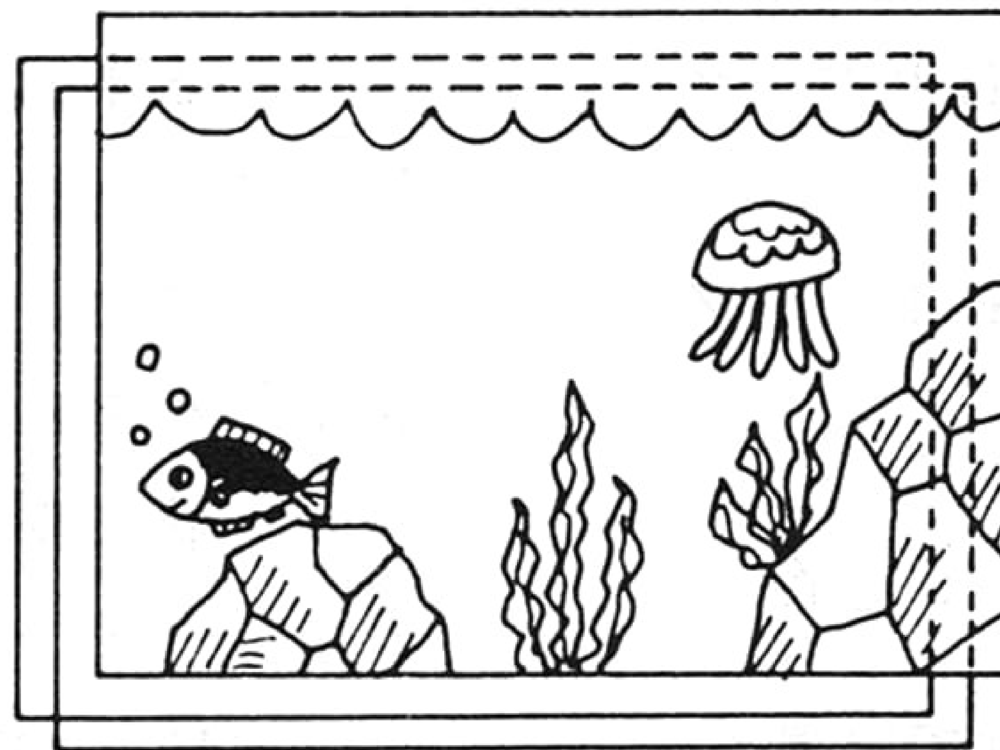
②次に透明なセルロイドの上に魚を一匹描きます。



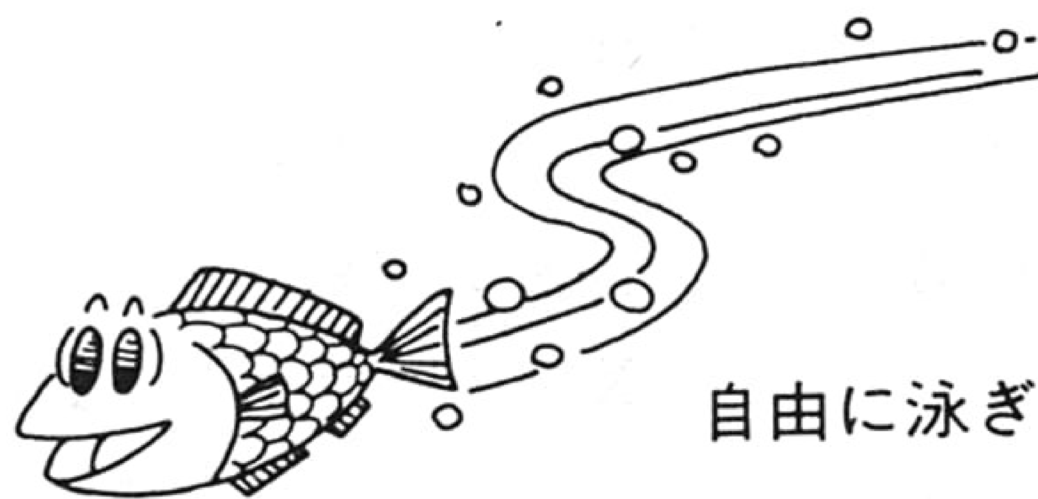
④画用紙の上に2枚のセルロイドを重ねてみます。



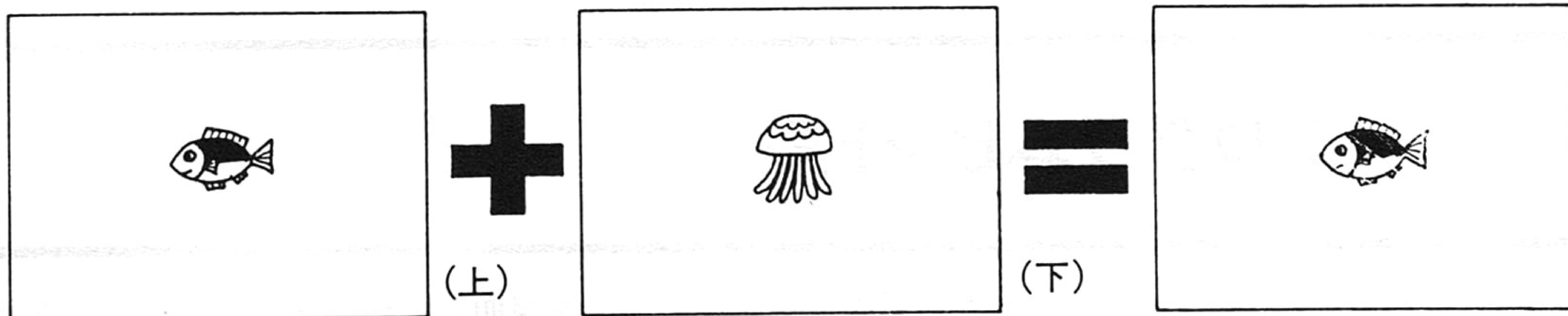
⑤セルロイドを上下、左右に動かすことにより、動物の位置が変わります。魚の絵が描かれたセルロイドを左側に動かせば魚が動いたようになります。



●スプライトキャラクターとは
セルロイド上に描かれた魚の絵や、クラゲの絵をスプライトキャラクターと呼びます。このスプライトキャラクターは背景の上を自由に動かせます。




●スプライト表示面
セルロイド上に魚などのスプライトキャラクターを描きましたが、このセルロイドがスプライト表示面になります。このスプライト表示面にはひとつのスプライトキャラクターを描くことができます。
スプライト表示面は32枚まで重ね合わせられます。魚の描かれたセルロイドとクラゲの描かれたセルロイドを重ね合わせたとき、魚とクラゲが同じ位置にあれば、上に乗せたセルロイドの絵に下のセルロイドの絵が隠れてしまいます。




スプライト表示面も重ね合わせ方によって下の絵が隠れてしまいます。上のセルロイドから順番に0、1、2、3〜31と番号をつけてスプライト表示面と呼びます。
スプライト表示面0とスプライト表示面1に魚とクラゲを描き、重なったときに魚を表示したいときは、魚をスプライト表示面0に描きます。表示面0が優先されるからです。

●……で結局、スプライトとは？
スプライトとは、背景の上にいろいろなキャラクターを自由に表示させることができる機能のことです。キャラクターが動いても、背景の絵がこわれたりはしません。つまり動画（絵が動く）です。

# プログラムヒント集

ここでは、プログラムを作っていく上でのテクニックやヒントを説明していきます。またプログラムの定石ともいえる、いくつかの基本的な処理をみてみましょう。

## 6-1 乱数を自由にあやつる

乱数を発生させるには、RND関数があります。このRND関数を使うことによって簡単に乱数を得ることができます。しかし、この乱数は0以上1未満なので、サイコロの目のように1から6までにはなりません。乱数の発生する数値をうまく利用して必要な数値を得ることを習いましょう。

### (1)乱数を発生

RND関数は引数が正のとき、値に関係なく次の乱数を与えます。引数がマイナスのときは、値により異なった乱数系列を与えます。(151P参照)

#### ① 処理ごとに同じ乱数を発生させたいとき

実際に乱数が必要な処理の前にRND関数を一度実行します。このときの引数は－1にします。これで、実行するごとに同じ系列の乱数が発生されます。(すなわち同じ乱数の並びになる。)

```
10 A=RND(-1)
20 FOR J=1 TO 5
30   PRINT RND(1)
40 NEXT J
50 END
run
 .0962486816692
 .2106965585230l
 .326517363050４
 .47775124336581
 .3409147084636
Ok
run
 .0962486816692
 .21069655852301
 .326517363050４
 .47775124336581
 .3409147084636
Ok
```

#### ② 処理ごとにちがう乱数を発生させたいとき

実際の処理の前に発生させる乱数をRND(－TIME)とします。TIMEは時間をもっているシステム変数で、処理するたびに時間が変化しているわけですから、そのたびにちがう乱数系列になり、発生する乱数が毎回ちがってきます。

```
10 A=RND(-TIME)
20 FOR J= 1 TO5
30   PRINT RND(1)
40 NEXT J
50 END
run
 .9340186816692
 .81332655852301
 .2914873630504
 .09718124336581
 .0250847084636
Ok
run
 .1649086816692
 .24523655852301
 .8527773630504
 6.69124336581E-03
 .4607747084636
Ok
```

※乱数ですから、上の表示になるとは限りません。

**問題**

変数をすべて整数式に定義し、乱数を8〜13まで発生させるプログラムを作りなさい。

**考え方**

8〜13という乱数は0〜5という乱数を発生させたあとに8をたします。

**プログラム例**

```
10 DEFINT A-Z
20 R=RND(-TIME)
30 FOR J=1 TO 10
40   A=RND(1)*6
50   A=A+8
60   PRINT A
70 NEXT J
80 END
```

## (2) 乱数の整数化

発生させる乱数を整数化するにはどうしたらよいでしょうか。たとえば0〜9までの範囲の乱数を発生したいときには、RND(1)の値が0〜0.99999999999999´ですからこの値に10をかければ、0〜9.9999999999999になります。そのあとに少数点以下を切り捨てればよいことになります。このときは、INT関数を使います。

A＝INT (RND(1)＊10) となります

しかし、発生した乱数を記憶する変数の型が整数型であれば、INT関数を使う必要はありません。

DEFINT A〜Z

↓

A＝RND(1)＊10

↓

では、発生する値を1〜10の範囲にしたいときは、どうすればいいでしょう。先ほどの例が0〜9でしたので、単純に＋1すればよいことになります。

A＝INT (RND(1)＊10) ＋1

**ワンポイント**

ユーザー定義関数であるDEF FN文を使って0から任意の数までの乱数を発生する関数を作ると、大変便利です。

DEF FNR(X)＝RND(1)＊X

と関数を定義すれば、プログラム中でFNR(10) とすれば0〜9までの乱数を発生します。FNR(5)とすれば当然0〜4になります。

```
10 DEFINT A-Z
20 R=RND(-TIME)
30 DEF FNR(X)=RND(1)*X
40 FOR J=1 TO 10
50   PRINT FNR(10)
60 NEXT J
70 END
```

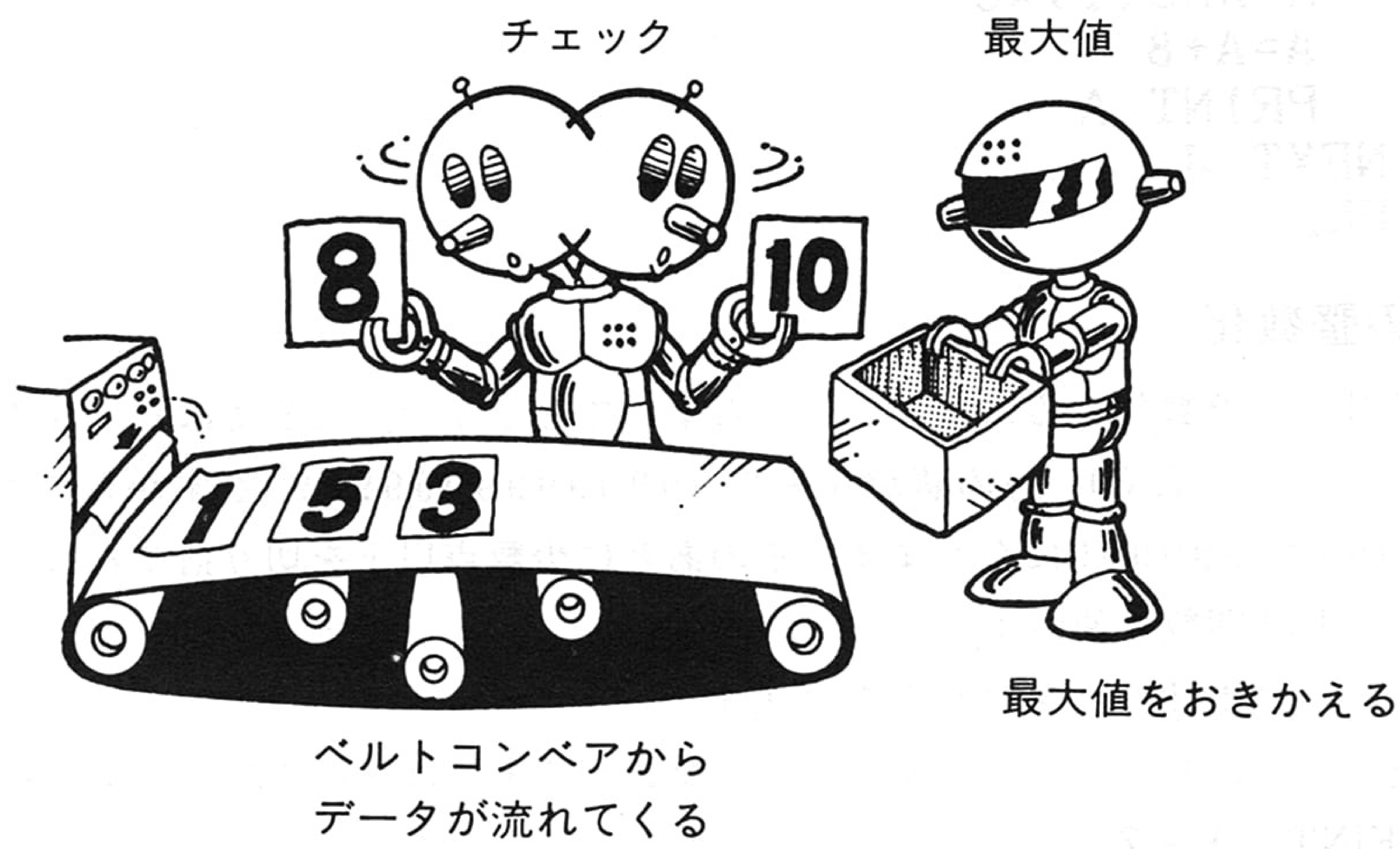
(DEF FN文に関しては141P参照)

## 6-2 最大値を求めるプログラムを考える

プログラムでデータの最大値を求めたり、最小値を求めたりすることがあります。ここでは、最大値を求める方法を考えてみます。

**プログラムの内容**

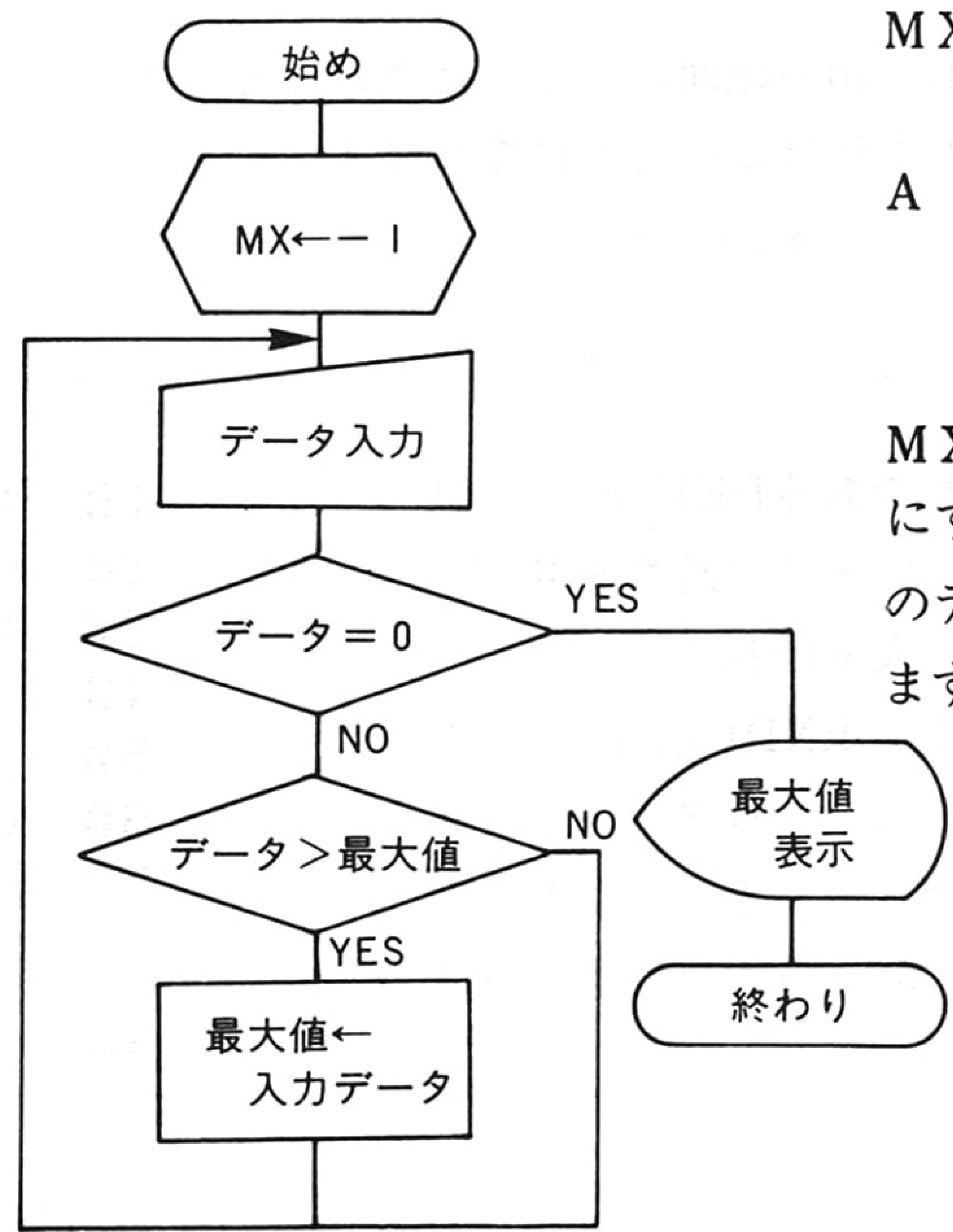
データをキーボードから入力していき、いま記憶している最大値と比較し、その最大値より大きいデータが入力されたら、そのデータを最大値におき換えます。
最初に記憶している最大値は、これから入力されるどのデータよりも小さい値にしておきます。
データが0のとき処理を終えます。




**プログラム**

```
10 DEFINT A-Z
20 MX=-1
30 INPUT "DATA:";A
40 IF A=0 THEN 80
50 IF A>MX THEN MX=A
60 GOTO 30
70 REM
80 PRINT "ｻｲﾀﾞｲﾁ";MX
90 END
```

**フローチャート**




**変数リスト**

MX：最大値を記憶する変数
A　：入力データ

MXの初期値をマイナスにすることにより、最初のデータが最大値になります。

# 6-3 入力データの合計を求める

データの合計を計算するときは、どのようにすればよいでしょう？

例）キーボードから入力されたデータの合計を求めます。入力データが0のとき、入力を終了し、合計を表示します。

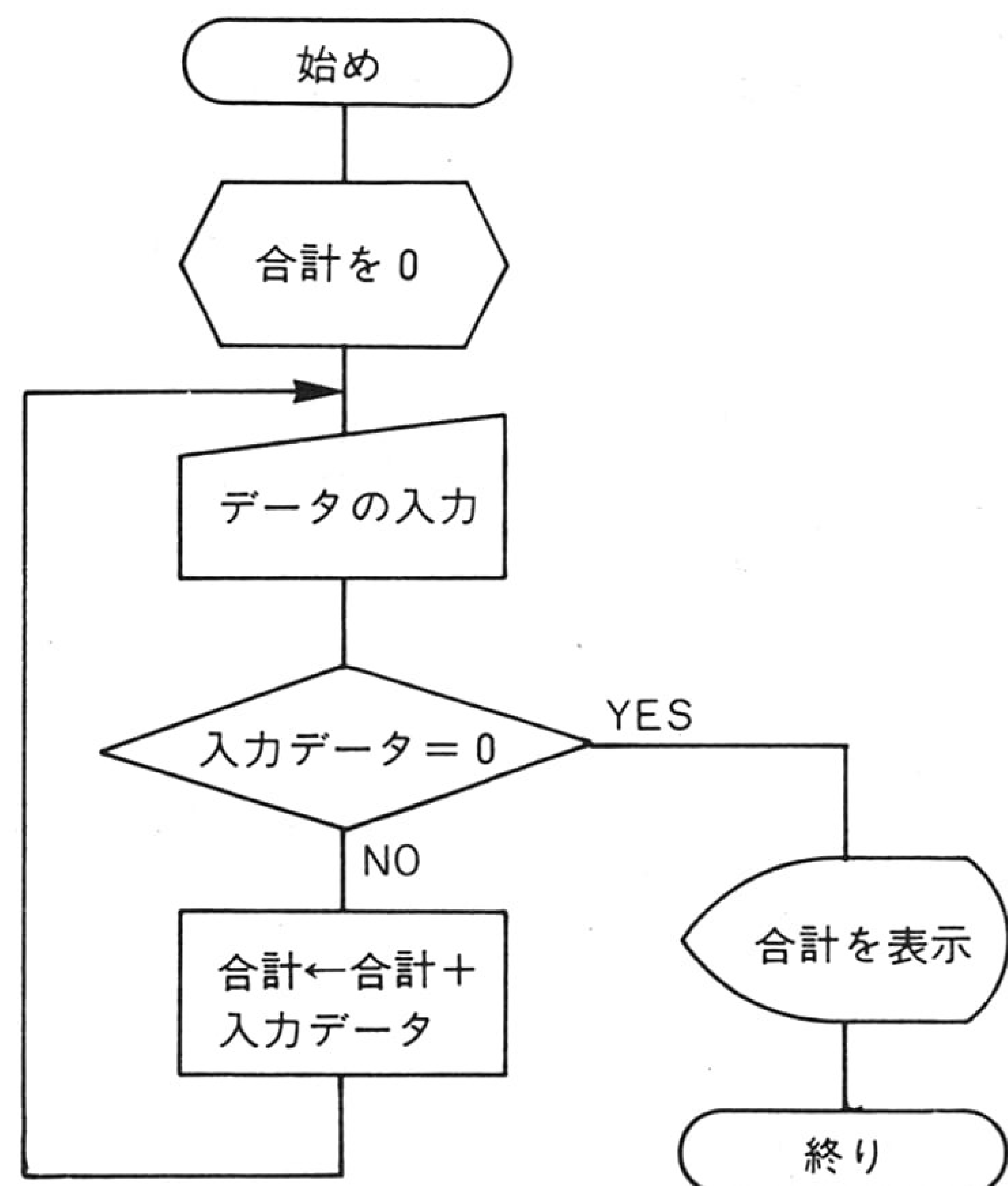
（合計）⇐ （合計）+（新しいデータ）

最初に合計を0にしておき、データが入力されるたびにそのデータを合計に加えていきます。

**プログラム**

```
10 SUM=0
20 INPUT"DATA:";DA
30 IF DA=0 THEN 70
40 SUM=SUM+DA
50 GOTO 20
60 REM
70 PRINT "ｺﾞｳｹｲ";SUM
80 END
```

**フローチャート**




変数リスト

SUM：合計を記憶する変数

DA：入力データ

# 6-4 グラフィック画面に文字を表示する方法

グラフィック画面には、通常絵を表示しますが、文字も表示することができます。テキスト画面に文字を表示するときは、LOCATE文とPRINT文を用いますが、グラフィック画面に文字を表示するときはLOCATE文やPRINT文は使用できません。

グラフィック画面に文字を表示するには次の手順を踏んでください。

①OPEN文でグラフファイルを開く（OPEN文）

②表示したいところに文字があるときは、その文字を消す（LINE文でBOX指定）

③表示位置を指定する（PSETまたはPRESET文）

④文字を表示する（PRINT＃文）

では、そのひとつひとつを説明していきましょう。

①OPEN文でグラフファイルを開く

この命令を実行すると、PRINT＃文が可能になります。OPEN文の書き方の詳細は「リファレンスブック」を参考にしてください。

ここでは、次の書き方だけを覚えておきましょう。

OPEN "GRP:" FOR OUTPUT AS ＃1

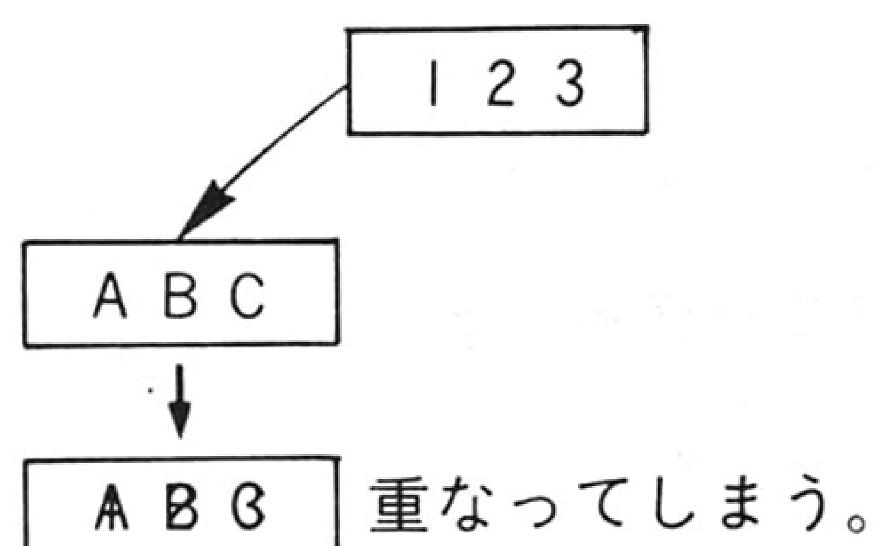
ここの数字をPRINT＃でも使います。↑

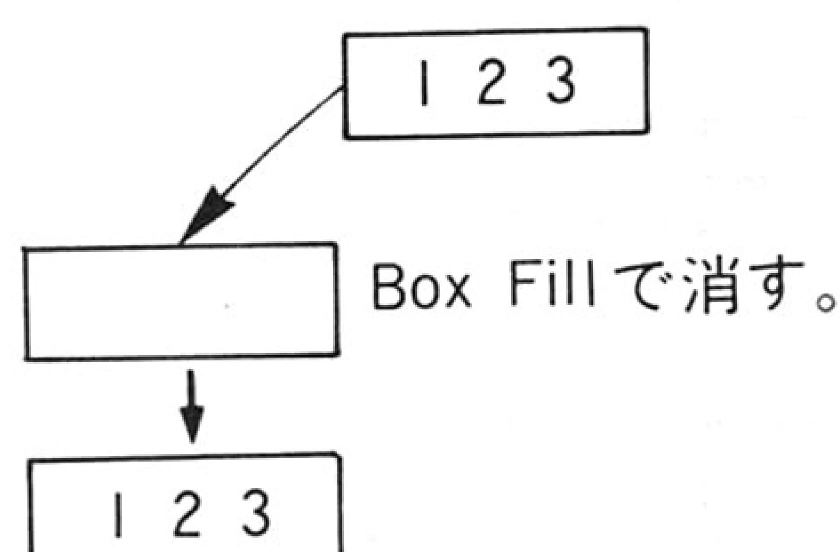
OPEN文はプログラムの最初に一度だけ実行します。2度実行すると "File already open" エラーになります。

②画面上に表示する文字のところを消す

グラフィック画面に文字がすでに表示してあるところに新たに文字を表示しようとすると文字が重なってしまいます。

ＡＢＣが表示されているところに1、2、3を表示すると次のようになってしまいます。

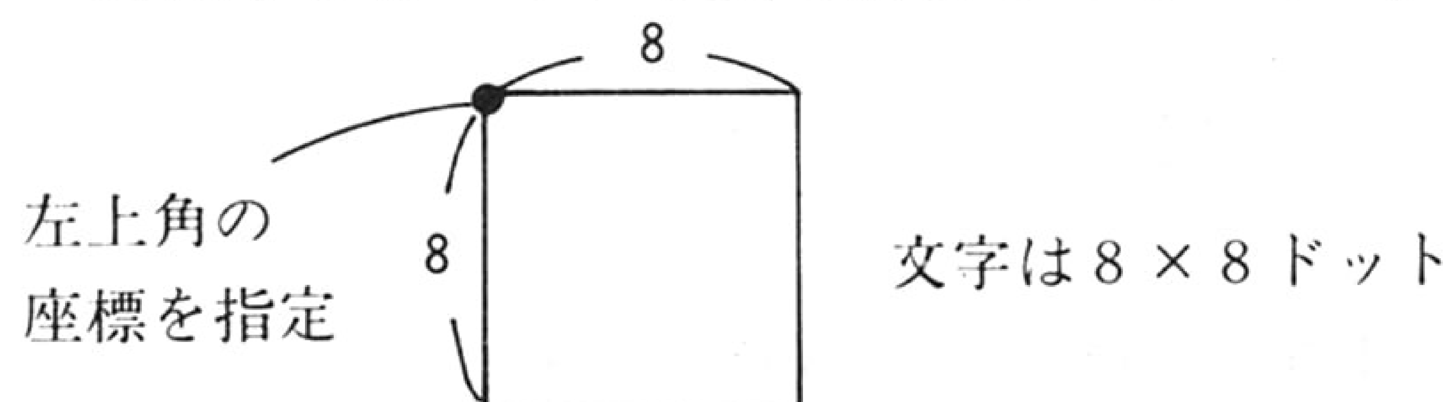



このため、すでに表示されている文字を消すには、LINE文でわくのなかを消します。

③表示位置を指定する

表示する文字の左上の角を画面上にセットします。セットの方法はPSET文かPRESET文を使います。画面の背景の色に合わせて点を表示しないと、点がでてしまいますので何色を使うとよいのか考えて指定してください。




文字を出す位置の色がCOLOR文の背景色と同じとき――PRESET(X座標，Y座標)

文字を出す位置の色がCOLOR文の背景色とちがうとき――PSET(X座標，Y座標)
色コード：文字を表示する位置にある色

**ワンポイント**

文字を表示するときの位置は、テキスト画面（SCREEN 1）と対応させて考えるようにすれば、文字の位置計算が非常に楽です。文字は８×８ドットで表示されますので、グラフィック画面の横の列に32文字（256／8）表示されます。グラフィック画面の０～７ドットを１文字目、８～15ドットを２文字目としていきます。文字表示の左はしの座標は次の計算で求めることができます。

X座標＝(表示桁－１)×８

同じようにY座標は次の計算式で求めることができます。

Y座標＝(表示桁－１)×８

④文字を表示する

文字を表示するには、PRINT#文で表示します。PRINT#の詳しいことは (218P参照)

PRINT#では、OPEN文で指定したファイル番号を指定します。OPEN文を実行しないでPRINT#を実行すると〝File not OPEN〞エラーになります。

では、まとめとしてサンプルプログラムを見てください。このプログラムは得点を表示するものです。

**プログラム**

```
10 COLOR 15,4,4
20 SCREEN 2
30 OPEN"GRP:" FOR OUTPUT AS #1
40 PT=0
50 PRESET(8*4-1,0)
60 PRINT #1,"POINT:"
70 FOR J=1TO 100
80      LINE(8*10-1,0)-(8*15-1,8-1),4,BF
90      PSET(8*10-1,0),4
100      PRINT#1,PT
110      PT=PT+10
120      FOR K=1 TO 300:NEXT K
130 NEXT J
140 END
```

| | |
|---|---|
| 10行 | 画面の色を設定 |
| 20行 | グラフィック画面を設定 |
| 30行 | OPENで画面出力ファイルをオープンする |
| 40行 | 得点を0にする |
| 50～60行 | 〝POINT:〞をLOCATE 8,0にあたる位置に表示 |
| 70行 | FOR文(80～120行をくり返し) |
| 80行 | 表示文字を消去 |
| 90～120行 | 得点の表示<br>時間待ち |
| 130行 | NEXT文 |
| 140行 | 終り |

**プログラムの改造**

80行のLINE文の色指定と90行のPSET文の色指定を1に変えてみてください。

かわったイメージになります。

# 第7章 プログラムを楽しもう

## 7-1 カラーテスト

MX-101は、テレビの画面に16色の色を表示することができます。どんな色が表示されるか試してみましょう。

使い方：プログラム入力後、F5か RUN↵ を押すと画面には16色の色が表示されます。終了するときは、何か別のキーを押せばいいのです。

```
10 SCREEN 2:COLOR 15,4,7
20 OPEN "GRP:" AS #1
30 FOR C=0 TO 15
40 READ A$
50 PSET(164,12*C+2),4
60 PRINT #1,A$
70 LINE(20,12*C)-(159,12*(C+1)-1),C,BF
80 NEXT C
90 A$=INPUT$(1)
100 DATA ﾄｳﾒｲ
110 DATA ｸﾛ
120 DATA ﾐﾄﾞﾘ
130 DATA ｱｶﾙｲ ﾐﾄﾞﾘ
140 DATA ｸﾗｲ ｱｵ
150 DATA ｱｶﾙｲ ｱｵ
160 DATA ｸﾗｲ ｱｶ
170 DATA ﾐｽﾞｲﾛ
180 DATA ｱｶ
190 DATA ｱｶﾙｲ ｱｶ
200 DATA ｷ
210 DATA ｱｶﾙｲ ｷ
220 DATA ｸﾗｲ ﾐﾄﾞﾘ
230 DATA ﾑﾗｻｷ
240 DATA ﾊｲ
250 DATA ｼﾛ
260 END
```

## 7-2 音楽テスト

まず8オクターブのドレミファソラシドを演奏し、そのあとで〝ドミソ〟〝ドファラ〟〝シレソ〟の3重和音を出してみます。
こうして、MX-101の音楽機能を試してください。

使い方：プログラム入力後、F5か RUN↵ を押します。8オクターブの音を出して、次に3重和音を出します。

```
10 FOR O=1 TO 8
20 PRINT O;" ｵｸﾀｰﾌﾞ"
30  PLAY "T64O"+STR$(O)+"CDEFGAB"
40 IF PLAY(1) THEN 40
50 NEXT O
60 PRINT"ﾜｵﾝ"
70 A$="T32O4CCO3B"
80 B$="T32O4EFD"
90 C$="T32O4GAG"
100 PLAY A$,B$,C$
110 IF PLAY(1) OR PLAY(2) OR PLAY(3) THEN 110
120 END
```

## 7-3 ジョイパッドテスト

MX-101にはジョイパッドがついています。これを使って画面上に線を描くことを試してみましょう。

```
10 COLOR 15,1,1
20 SCREEN 2
30 X=255/2:Y=190/2:P=1
40 PSET(X,Y)
50 IF P THEN PRESET(X,Y)
60 C=STRIG(1):IF C THEN P=0
70 D=STRIG(3)
80 IF D THEN 230
90 A=STICK(1)
100 IF A=1 THEN Y=Y-1
110 IF A=2 THEN X=X+1:Y=Y-1
120 IF A=3 THEN X=X+1
130 IF A=4 THEN X=X+1:Y=Y+1
140 IF A=5 THEN Y=Y+1
150 IF A=6 THEN X=X-1:Y=Y+1
160 IF A=7 THEN X=X-1
170 IF A=8 THEN X=X-1:Y=Y-1
180 IF X<0 THEN X=0
190 IF Y<0 THEN Y=0
200 IF X>255 THEN Y=255
210 IF Y>191 THEN Y=191
220 GOTO 40
230 COLOR 15,4,7
240 END
```

使い方：プログラム入力後、F5か RUN↵ を押すと画面の中央に白い点がチカチカします。ジョイパッドを押すと、この点が動きますが、点滅しているうちは線は描けません。
TR1を押すと、点滅がとまりますので、これでジョイパッドを動かすと線が描けます。終了するときはTR2を押します。

## 7-4 スプライトテスト

MX-101の特長であるスプライトを使い、画面上のキャラクターを動かしてみましょう。

```
10 COLOR 15,4,7
20 SCREEN 1,1
30 FOR J=1 TO 8
40   READ A$:F$=F$+CHR$(VAL("&H"+A$))
50 NEXT J
60 SPRITE$(1)=F$
70 FOR X=0 TO 255
80   PUT SPRITE 1,(X,10),15,1
90 NEXT X
100 END
110 DATA 38,38,13,7C
120 DATA B8,38,68,4C
```

使い方：プログラム入力後、F5か RUN↵ を押すと、人間のキャラクターが左から右へ走っていきます。

## 7-5 塗りつぶしテスト

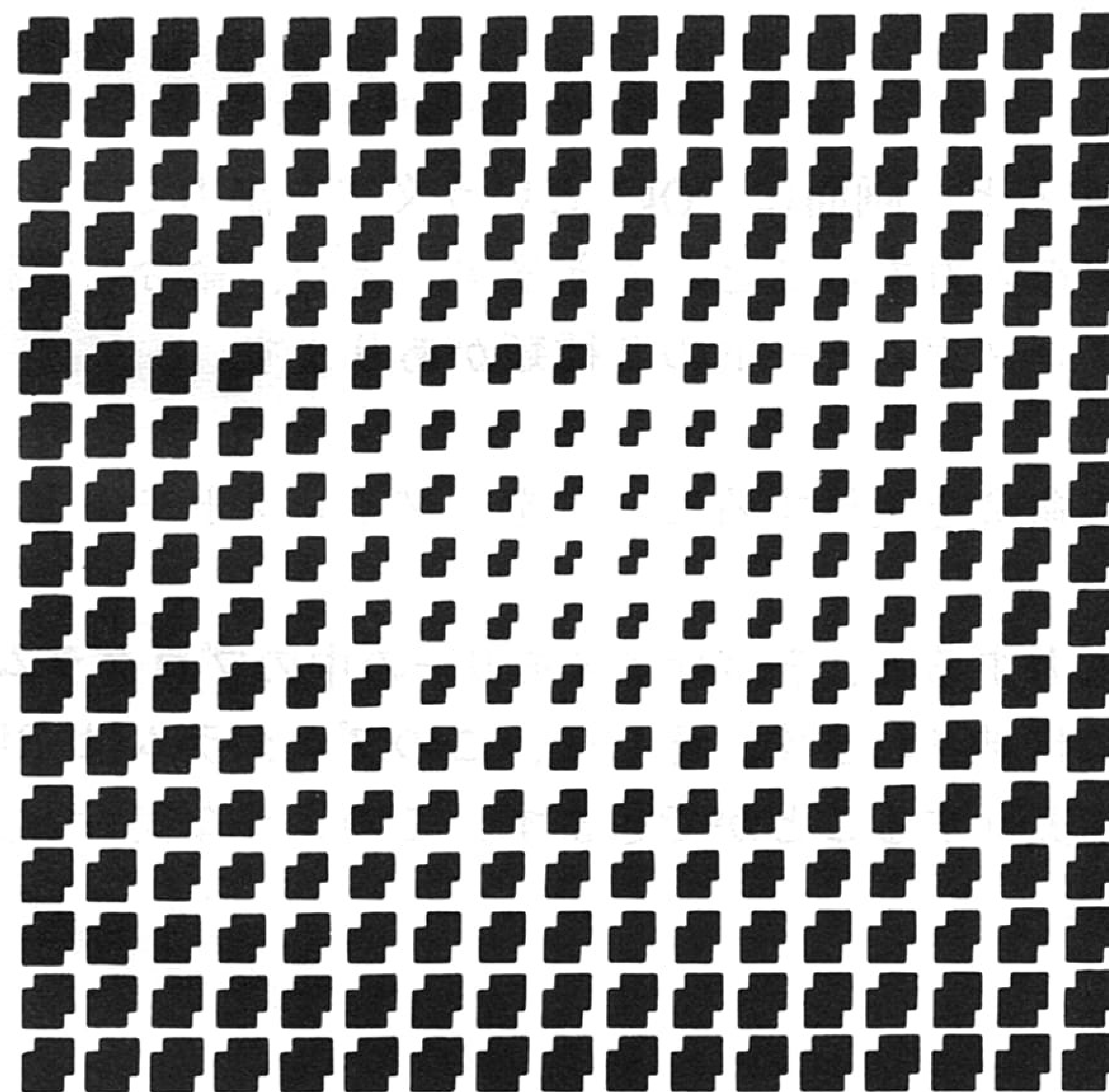
乱数を使って画面に四角を描き、乱数で色を決め、四角形のなかを塗りつぶします。

```
10 COLOR 15,0,0
20 SCREEN 2
30 DEFINT A-Z
40 DEFFNR(X)=X*RND(1)
50 D=RND(-TIME)
60 REM
70 FOR J=1 TO 100
80 X1=FNR(128):Y1=FNR(96)
90 X2=FNR(128):Y2=FNR(96)
100 C=FNR(16)
110 LINE(X1,Y1)-(X2,Y2),C,BF
120 REM
130 X1=FNR(128):Y1=FNR(96)
140 X2=FNR(128):Y2=FNR(96)
150 LINE(X1+128,Y1+96)-(X2+128,Y2+96),15,B
160 NEXT J
170 COLOR 15,4,7
180 END
```

**使い方**：プログラム入力後、F5 か R U N ↵ を押します。画面に四角形があらわれるでしょう。画面の左¼は塗りつぶしの四角形で、右下は塗りつぶされない四角形のはずです。
塗りつぶされた四角形、塗りつぶされない四角形がそれぞれ100個ずつ描かれたら終了します。

# 第3部

# リファレンス



# 第1章
# MSX BASIC

## 1-1 概要

MSX BASICは、マイクロソフト社のBASICを拡張した機種で、ホームユースからビジネスまで多彩に使うことができます。また、MSX規格のパーソナルコンピュータは共通の仕様になっています。

## 1-2 動作モード

MSX BASICが起動すると、画面に〝Ok〟という文字を出力し、キー入力待ちになります。この状態のとき、文字を入力することができます。また、命令を実行する方法として、ダイレクトモードと、プログラムモードの2種類があります。
〔ダイレクトモード〕
行番号をつけずに直接命令を実行するモードを、ダイレクトモードといいます。
〔プログラムモード〕
行番号をつけて行を入力すると、その行はメモリーの中のプログラムの1行となり、行番号とともにメモリー上に格納されます。そして、このプログラムはRUNコマンド、GOTO文、GOSUB文により実行することができます。このような実行方法を、プログラムモードによる実行と呼びます。

## 1-3 文と行

文とは、BASICが行なう手続きを記述している最小単位です。これには、コマンド、ステートメント、および代入を行なう式を含むことができます。
また、文はコロン(:)を用いて他の文とつなぐこともできます。これはマルチステートメントと呼ばれ、行番号も含めて1行の長さは255文字以内です。
行は、BASICに対して命令をするときの単位で、キー入力を開始してから、リターンキーを押すまでを意味します。1行の最大は255文字です。
〔コマンド〕
コマンドは、LOADやRUNのような、コンピュータを直接制御するための命令のことをいいます。コマンドは、通常ダイレクトモードで使用します。

〔ステートメント〕
ひとつの仕事をさせる命令文のことです。たとえば計算をしたり、画面に文字をだしたり、線を引いたり、音楽を演奏したりする命令です。

## 1-4 行番号

多くの行を使ってプログラムしたい場合は、行の先頭に行番号を入れます。行番号のついた行を入力すると、メモリー上にプログラムを構成する行として、メモリー上に格納されます。行番号は0～65529の整数で、メモリーに納められる順序になります。プログラムの実行は行番号の若い（小さい番号順）方から行なわれます。また行番号は、分岐や編集の目じるしにもなります。

## 1-5 使用できる文字

MSX BASICでは、英文字（大文字、小文字）、数字、ひらがな、カタカナ、特殊記号、およびグラフィック文字を使うことができます。文字は付録（249P）の「キャラクターコード表」を参照してください。

## 1-6 特殊記号の使い方

MSX BASICでは、演算子（+，－，＊，／）などのほかに、特別な意味をもつ特殊記号があります。

.（ピリオド）　現在、MSX BASICが着目している行番号を示します。またプログラム実行中にエラーが発生したときや、スクリーンエディタで行を挿入したときにその行を記憶しています。さらにLIST, RENUM文などでは行番号の代りに使うこともできます。

〔例〕 LIST.

－（ハイフン）　LIST, DELETEコマンドなど、行の範囲（何行めから何行め）を指定する場合に使うことができます。

〔例〕 DELETE 200－300

：（コロン）　マルチステートメントの区切りとして使います。

〔例〕 A=10+7:PRINT A

，（カンマ）　PRINT, INPUT文など、パラメータが並ぶ場合の区切りとして使います。

〔例〕 INPUT A,B
　　　COLOR 5,6,7

；（セミコロン）　PRINT文などの区切りとして使います。

〔例〕 PRINT"ANS=";A

'（アポストロフィ）　REM文（注釈文）の代りに使います。

〔例〕 PRINT A'コタエノヒョウジ

（スペース…空白）　プログラムを見やすくするために、プログラム中に自由にスペース（空白）を入れることができます。プログラム中の空白は、命令の実行中は無視されますが、メモリー上に格納するときは、一緒に格納されます。コマンド、ステートメント、関数、システム変

数、行番号の中には、スペースを入れることはできません。

?(疑問符)　　PRINT 文の代用として使用できます。
〔例〕 ?"HELLO"

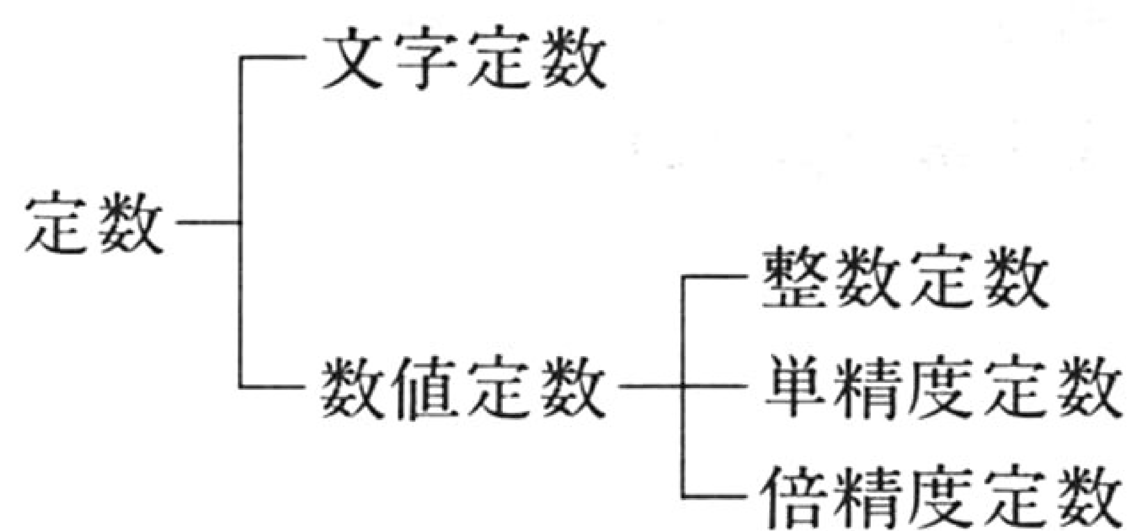
## 1-7 コントロールキーおよびコード

MSX BASICでは、キーボード上に書かれた、キャラクター文字以外に特別な意味を持つコントロールキーおよびコードがあります。これについては付録「コントロールコード表」(248P) を参照してください。

## 1-8 定数

**●定数の種類**

MSX BASICの定数には次の種類があります。




**●文字定数**

文字定数とは、ダブルクォーテーション(")で囲まれた255文字以内の文字列で、数字、英文字、カタカナ、ひらがな、記号などを含むことができます。

〔例〕 "1234"　"ABCDEF"

**●数値定数**

数値定数は、たし算やひき算、算術演算などをすることができる定数で、数値の表わせる範囲で整数、単精度、倍精度に分かれます。

**●整数定数**

−32768から＋32767までの整数です。ただし、小数点を含むことはできません。整数計算は、非常に高速にできます。

**●単精度定数**

6桁以内の実数定数、あるいは6桁以内の整数とEを使った指数表示の組み合わせです。Eは−64から＋62までの範囲をとることができます。
数値の最後に！をつけた数値も単精度定数です。
数値が6桁を越えると7桁目が四捨五入されます。

〔例〕 1.23　65000　3.14!　5.76E+15($5.76 \times 10^{15}$)

**●倍精度定数**

単精度よりもさらに精度の高い数値です。7桁以上の実数、整数とEを使った指数表示の組み合わせです。Eの範囲は−64から＋62までで、EのかわりにDを使うこともできます。また、数値の最後に#をつけた数値も倍精度とみなします。14桁をこえる数値は15桁目が四捨五入されます。

〔例〕 12345678　123#　1.23D−64($1.23 \times 10^{-64}$)　0.9999999999

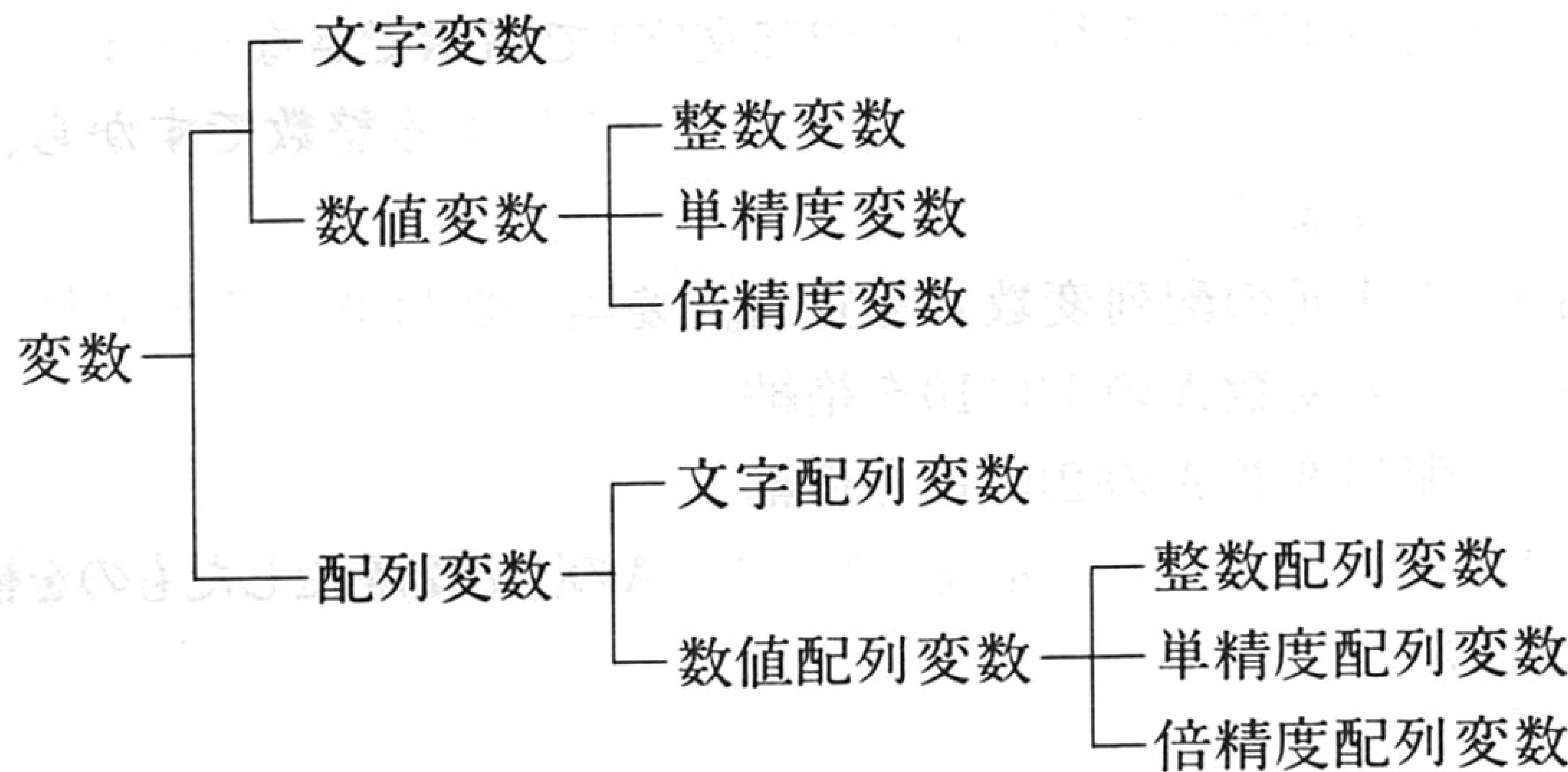
## 1-9 変数

### ●変数とは

プログラム中で、文字を格納したり、数値を格納したりする記憶エリアです。

### ●変数の種類

変数は、取扱う数値、文字により種類が分かれています。




### ●変数名

英文字から始まる2文字を変数名とし、先頭の1文字は、英字でなければなりませんが、2文字目は数字でもかまいません。3文字以上の変数名は3文字目からは無視され、最初の2文字までしか判別されません。また、変数中に含まれるスペースや、カタカナ、グラフィック文字は無視されます。

なお、変数名は予約語と同じであったり、予約語を含んでいたりしてはいけません。小文字と大文字の区別はなく，すべて大文字になります。予約語については、付録の「予約語」(251P) を参照してください。

〔例〕 ・良い例

HE

H5

HS

Kコタエ0 (変数名はKOとされる)

KOTAE (変数名はKOとされる)

・悪い例

7I (先頭の文字が英字でない)

BASEX (予約語のBASEが含まれている)

### ●変数の型

変数は格納したいデータの型に応じて区別することができます。変数の型は、変数名の最後に、型宣言文字をつけることによって区別しますが、型宣言をしない変数は、すべて倍精度型変数とみられます。型宣言文字は次の通りです。

%……整数型　(例)　AB%　SE%

!……単精度型　(例)　TA!　KO!

#……倍精度型　(例)　PAI#　ANS#　AB

$……文字型　(例)　NAMAE$　JUSHO$

なお、変数の型は、DEF INTなど型宣言文を使うことにより定義することもできます。

### ●配列変数

配列変数は、文字変数または数値変数に添字をつけて表示します。また、配列変数はいくつかの要素で構成され、その次元と要素の数はDIM文であらかじめ宣言しておきます。

3次元までの配列変数については、宣言なしで用いることができ、その場合添字は最大10とみなされます。配列変数に含まれるデータを代入したり、参照するときは、添字を使って何番目の要素かを指定します。

配列の次元は理論上は255次元（実際は1行の長さ(255文字)で指示できない）まで、また添字はメモリーの不足しない限り許されます。添字は0から始まる整数ですから、配列要素は添字の最大値＋1となります。

〔例〕
```
10 DIM A(5)            1次元の配列変数Aを宣言。(要素の数は0～5の6個)
20 A(1)=10             配列変数Aの(1)に10を格納
30 A(2)=15             配列変数Aの(2)に15を格納
40 A(3)=A(1)+A(2)      配列変数のAの(3)にAの(1)と(2)をたしたものを格納
50 PRINT A(3)
```

| A(0) | A(1) | A(2) | A(3) | A(4) | A(5) |
|---|---|---|---|---|---|
| | 10 | 15 | 10+15 | | |

### ●変数とメモリー容量

変数は定義された直後は数値の場合0、文字の場合〝〟(ヌルストリング：何もない)が格納されます。また、そのときのメモリー上に確保される領域はそれぞれ次のようになります。

| | | |
|---|---|---|
| 変数 | 整数型 | 5バイト |
| | 単精度実数型 | 7バイト |
| | 倍精度実数型 | 11バイト |
| | 文字型 | 6＋(文字列の文字数)バイト |
| 配列変数 | 整数型 | 5＋2×(要素数)＋2×(次元数)＋1バイト |
| | 単精度実数型 | 5＋4×(要素数)＋2×(次元数)＋1バイト |
| | 倍精度実数型 | 5＋8×(要素数)＋2×(次元数)＋1バイト |
| | 文字型 | 5＋3×(要素数)＋2×(次元数)＋1＋(各要素の文字列の総文字数)バイト |

### ●システム変数

MSX BASICは、BASIC自身であらかじめ持っている変数があります。これをシステム変数と呼びますが、これには次のような種類があります。

| | |
|---|---|
| BASE(n) | 画面出力に関するテーブルのアドレスを持っています。 |
| ERR | エラーが発生したときのエラーコードを持っています。この変数に値を代入することはできません。 |
| ERL | エラーが発生したときの行番号をもっています。この変数もやはり代入はできません。 |
| SPRITE$(n) | スプライトパターンを記憶します。 |
| TIME | インターバルタイマーと呼び1/60秒ごとに値が1増えます。この変数は0～65535の値を代入することができます。 |

| | |
|---|---|
| CSRLIN | カーソルの垂直方向の位置を記憶します。 |
| POS | カーソルの水平方向の位置を記憶します。 |
| LPOS | プリンタヘッドの水平方向を持っています。 |
| VDP | MSX の画面を制御する VDP のレジスタを操作します。 |

## 1-10 式と演算

式とは、定数や変数を演算子で結合した数式や、単に文字や数値、または変数だけのものをいいます。

MSX BASIC の演算は5つに分類できます。

- 算術演算
- 比較演算
- 論理演算
- 関数
- 文字列演算

### ●算術演算

算術演算子には、次のようなものがあります。

↓ 実行順

| 演算子 | 演　算 | 例 |
|---|---|---|
| ∧ | 指数演算 | X∧Y |
| — | 負　号 | −X |
| *，／ | 乗算，除算 | X*Y，X／Y |
| +，− | 加算，減算 | X+Y，X−Y |

演算の実行順序は上から優先順位の高いものとなります。優先順位を変更したいときはカッコ（　）を用います。カッコ内の演算子は、他の演算子よりも先に実行します。

次に代数表記を MSX BASIC の表記に直した例を示します。

| 代数表記 | MSX BASIC |
|---|---|
| $2X+Y$ | 2*X+Y |
| $\frac{X}{Y}+2$ | X／Y+2 |
| $\frac{X+Y}{2}$ | (X+Y)／2 |
| $X^2+2X+1$ | X∧2+2*X+1 |
| $X^{Y^2}$ | X∧(Y∧2) |
| $(X^Y)^2$ | X∧Y∧2 |
| $Y(-X)$ | Y*−X |

整数の除算は、／の代りに¥によって行ないます。ただし、扱う数値が実数の場合は、演算が実行される前に、小数点以下が切り捨てられます。商は、小数点以下が切り捨て

られた整数となります。

〔例〕 17¥3=5　(17/3=5…2)　BASICではPRINT17¥3とします。

　　　17.5¥5=3　(17/5=3…2)　BASICではPRINT17.5¥5とします。

剰余の計算はMODによって行なわれます。扱う数値が実数のときは、演算の実行前に小数点以下が切り捨てられます。結果は整数の割り算の余りになります。

〔例〕 13.5 MOD4=1　BASICではPRINT13.5 MOD4とします。

　　　7.6 MOD6.1=1　BASICではPRINT7.6 MOD6.1とします。

数値を0で除算した場合は、"Divison by zero"というエラーメッセージが出力されたあとにプログラムが中断され、コマンドレベルに戻ります。0に対して負になるべき乗を行なったときも同じです。

〔例〕 print 0 ∧ − 3

　　　Division by zero

代入や演算の結果が、その変数の型の範囲を越えると"桁あふれ"を生じ"Overflow"というエラーメッセージを出力したあとに、プログラムが中断しコマンド待ちに戻ります。

**●比較演算子**

比較演算子は2つの数値あるいは2つの文字列を比較するときに用います。結果は真のときには−1、偽のときは0となります。これはIF文で流れをかえるときなどに用います。

| 比較演算子 | 内　容 | 例 |
|---|---|---|
| = | (左辺) と (右辺) が等しい | X=Y |
| <>, >< | (左辺) と (右辺) が等しくない | X<>Y, X><Y |
| < | (左辺) は (右辺) より小さい | X<Y |
| > | (左辺) は (右辺) より大きい | X>Y |
| <=, =< | (左辺) は (右辺) より小さいか等しい | X<=Y |
| >=, => | (左辺) は (右辺) より大きいか等しい | X>=Y |

**●論理演算**

論理演算子は、複数の条件を調べたり、論理演算を行なうときに使います。論理演算はビットごとに0または1を結果として得ることができます。

①NOT = not (否定)

| X | NOT X |
|---|---|
| 1 | 0 |
| 0 | 1 |

② AND ＝ and（論理積）

| X | Y | X AND Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

③ OR ＝ or（論理和）

| X | Y | X OR Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

④ XOR ＝ exclusive or
（排他的論理和）

| X | Y | X XOR Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

⑤ EQV ＝ equivalance（同値）

| X | Y | X EQV Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

⑥ IMP ＝ implication（包含）

| X | Y | X IMP Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

条件判断文（IF 文）では、論理演算子を使って、複数の条件を判断することができます。

〔例〕
```
IF X<0 OR X>99 THEN～
IF X=0 AND Y=0 THEN～
IF NOT (X=0) THEN～
```

論理演算では、演算の前に扱う数値を －32768 から ＋32767 までの 2 の補数表現の整数

に変換します。もし、この範囲を越えると、“Overflow”エラーになります。

**● 関数**

関数とは、与えられたある引数に対して、決まった演算を行うもので、この演算の結果を値としてかえします。

MSX BASICは、“組み込み関数”として SIN（正弦）や SQR（平方根）のような数値関数や CHR$、LEFT$ などの文字列操作関数をあらかじめ用意しています。

また、ユーザーが自由に定義できる“ユーザー定義関数機能”も、もっています（DEF FN文）。

**● 文字列演算**

文字列演算は〝＋〟によって連結することができます。

〔例〕

```
A$="MSX"
B$="BASIC"
C$=A$+B$
PRINT C$
```

とすれば C$ には〝MSX BASIC〟が格納されます。

また、文字列の場合も数値と同じ関係演算子を使って文字列の比較を行なうことができます。文字列の比較は、文字列の最初から1文字ずつ、キャラクターコード（249P）の大きさで比較していきます。大きさはそれぞれの文字に対応するキャラクターコードの数値で表わされます。たとえば〝A〟と〝B〟はそれぞれ65と66になり、〝B〟の方が〝A〟より大きいといいます。このとき、もし、同じ文字列のときは等しくなり、片方が短かくて途中で比較が終ったときは、短い方が小さくなります。ただし、途中で1文字でも違ったときは、その時点で比較をやめ、キャラクターコードにより、大きいか小さいかを決めます。なお、文字列の比較では、スペース（空白）も意味をもちます。

**● 演算の優先順位**

1. カッコで囲まれたもの
2. 関数
3. 指数（べき乗）
4. 負号（−）
5. ＊，／
6. ¥
7. MOD
8. ＋，−
9. 関係演算子（<，>，＝など）
10. NOT
11. AND
12. OR
13. XOR
14. EQV
15. IMP

## 1-11 数値について

MSX BACICでは、標準での数値計算は定数の種類に関係なくすべて倍精度で行ないますので、単精度の計算も、いったん倍精度に変換されて計算され、再び単精度に変換されます。

〔例〕 A!＝5．375!／4．219!　実際は倍精度演算がなされます。

整数型は10進数、8進数、16進数、2進数で表記することができます。

- **10進数**
  −32768～32767で負の場合、数値の前に必ず負の符号(−)をつけなければなりません。ただし、正の場合は符号(＋)は省略できます。
- **8進数**
  先頭に&Oをつけた0～7までの数字の並びで&O 0～&O 177777の範囲をとることができます。
- **16進数**
  先頭に&Hをつけた0～9、A、B、C、D、E、Fまでの並び、&H0～&HFFFFまでの範囲をとることができます。&HFFFF～&H8000までは負の数で−1から−32768に対応します。
- **2進数**
  先頭に&Bをつけた0と1の並びで、範囲は&B 0～&B 1111111111111111ですが、&B 1111111111111111～&B 1000000000000000までは負の数の−1から−32768に対応します。2進数の数値を使った式では、演算子と2進数の間にスペース（空白）を入れないようにしてください。

## 1-12 型変換

数値は、データの型を必要に応じて他の型に変換することができますが、これはあくまでも数値間のことで、文字列と数値の間の変換はできません。型の変換は次の規則に従って行なわれます。

- 数値データを異なる型の数値変数に代入するときは、代入先の変数の型に変換されます。
  即ち

〔例〕 A%＝3.14　　　3.14をA%という整数型変数に代入する。
　　　PRINT A%　　　A%に格納されている3を表示する。

- 精度の異なる数値を使った演算では、精度の低い方の数値が、精度の高い方の型に変換されて演算が行なわれます。
- 論理演算はすべて整数型に変換され、その結果も整数で得られます。
- 実数が整数に変換されるときは、小数点以下が切り捨てられます。このとき、切り捨てた結果が、整数型の範囲（−32768～32767）を越えたときは“Overflow”エラーとなります。
- 倍精度実数型変数を単精度実数型変数に代入した場合、変数の値は、有効数字6桁にまとめたものとなります。これは単精度実数型の精度が6桁であり、もとの倍精度実数型の7桁目以降が四捨五入されて6桁となるのです。

## 1-13 エラーメッセージ

プログラムの実行を中断しなければならないようなエラーが実行時に発生したときは、エラーメッセージが画面に表示され、コマンドレベルに戻ります。

ダイレクトモードのエラーメッセージは

XXX……

プログラムモードでは

XXX……in yyyy

XXX……はエラーメッセージで、yyyyはエラーが発生した行番号です。なお、エラーについては、付録の「エラーメッセージ表」(252P)に詳しく、エラーの内容が書かれています。

※ただし、BREAK in yyyyはエラーメッセージではありません。

## 1-14 画面モード

MSX BASICで取扱われる画面は、大きくわけて、テキスト画面とグラフィック画面に分けることができます。テキスト画面では、リストなどの文字を出すことはできますが、点や線を描くことはできません。一方、グラフィック画面は、点や線などの絵を描くことはできますが、PRINT文で文字を出力することができません。グラフィック画面に文字を出す方法は、通常のPRINT文ではなく、グラフィック画面ファイル（GRP：）に対する出力の方法で文字を出します。

MSX BASICの画面モードには次の4種類があり、SCREEN文で選択できます。
電源投入時はテキストモード（32×24）の状態となっています。

| モード | 解像度 | 文字サイズ | 表示文字数 | スプライト | 備考 |
|---|---|---|---|---|---|
| テキストモード | —— | 6×8ドット | 最大40×24文字 | 不可 | 背景色と周辺色の区別ができない。 |
| テキストモード | —— | 8×8ドット | 最大32×24文字 | 可 | 電源投入時のモード |
| 高解像度グラフィックモード | 256×192ドット | —— | —— | 可 | 横8ドットを1単位とする領域に2色まで使える。 |
| マルチカラーモード | 64×48ブロック | —— | —— | 可 | ブロックごとに色指定できる。 |

※テキストモードでファンクションキーの表示をすると縦は23行になります。

● **テキストモード**（最大40×24文字）

このモードではスプライト機能やグラフィック命令を使うことができませんが、最大で、横40桁、縦24行の文字表示が可能です。(WIDTH文による指定がないときは、横39桁に設定されています。）文字の色も背景色も選択できます。

● **テキストモード**（最大32×24文字）

最大32桁、縦24行の表示ができます。(WIDTH文による指定がないときは、横29桁に設定されています。）表示文字は横8ドット×縦8ドットです。スプライト機能は使えますが、グラフィック命令は使うことができません。

● **高解像度グラフィックモード**

256×192ドットの解像度のグラフィックモードで、解像度が高いので、細かい点や線を描くことができます。横方向の8ドットを1単位とする領域ごとに2色までの色を使うことができます。このモードで文字を出力するときは、OPEN文で、"GRP："ファイルをオープンしたあとに、PRINT＃文で文字を出力します。

●マルチカラーモード

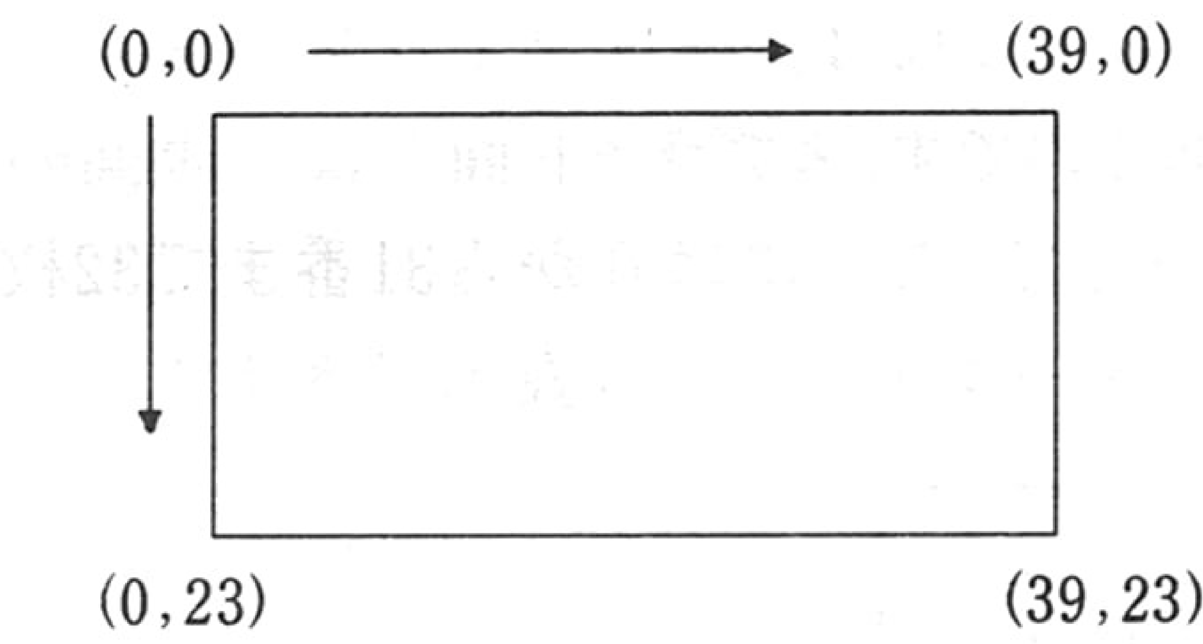
64×48ブロックの解像度のグラフィック画面、点を描く最小単位が４×４ドットのブロック単位なので、こまかい点や線を使うことはできませんが、１ブロックごとに色を使いわけることができます。

BASICがコマンドレベルに戻ると高解像度グラフィックモードのときも、マルチカラーモードのときも自動的にテキストモードに戻ります。

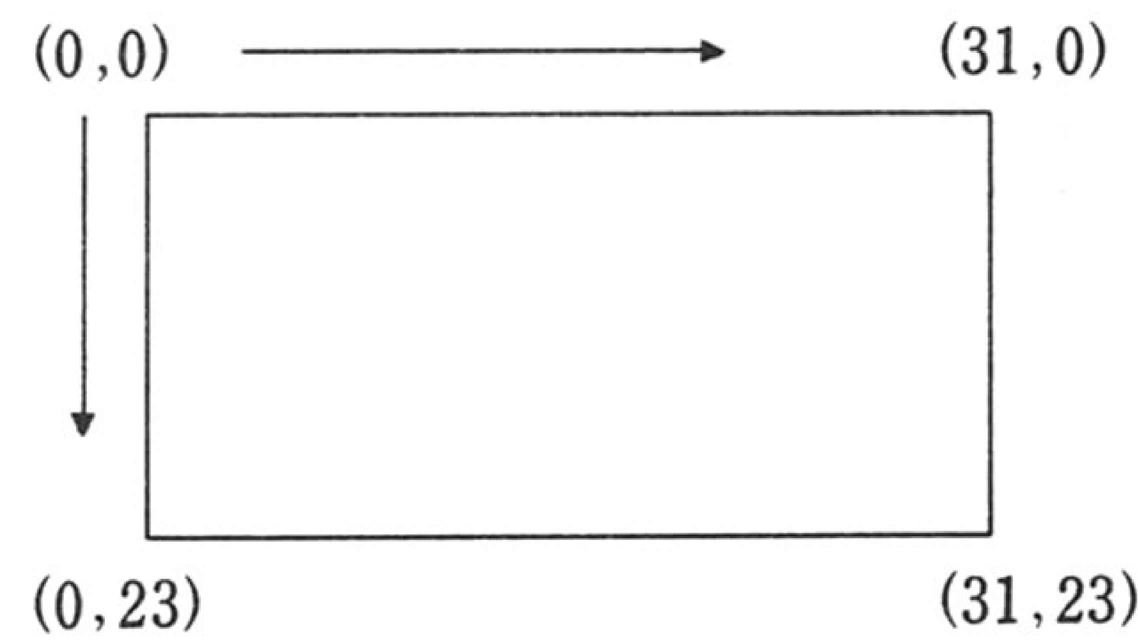
## 1-15 座標の指定

座標は、いずれも上端を（0,0）とします。

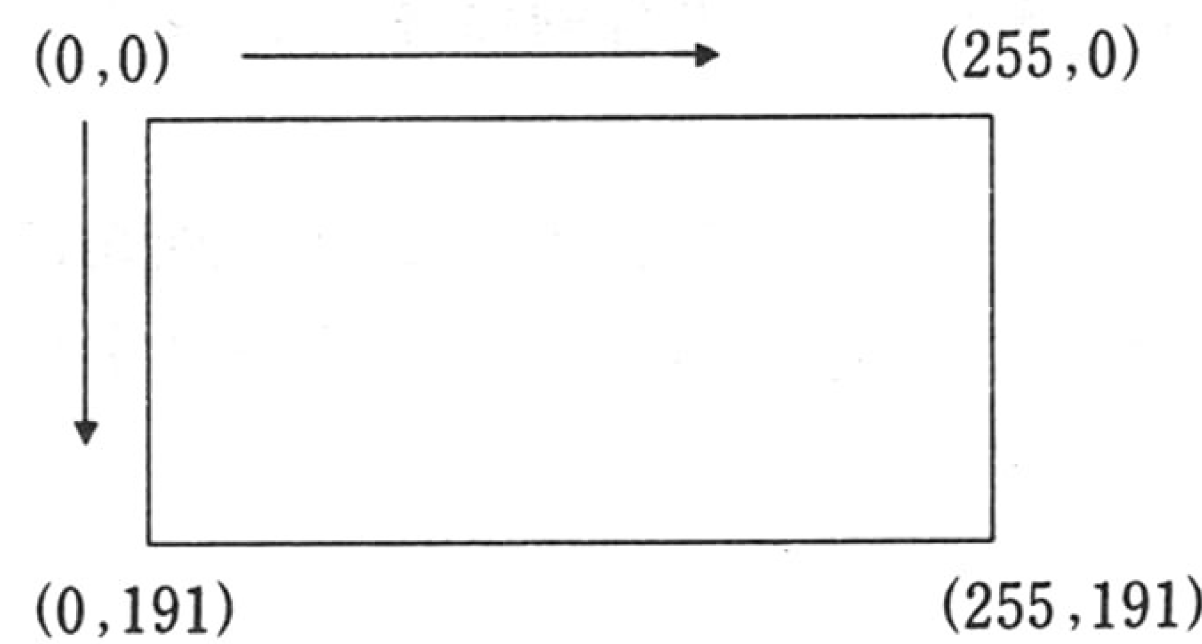
・テキストモード（最大40×24文字）




・テキストモード（最大32×24文字）




・高解像度グラフィックモード／マルチカラーモード




## 1-16 カラー

MSX BASICでは、次の16色を使うことができます。色はそれぞれカラーコードで指定します。

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 透明 | 4 | 暗い青 | 8 | 赤 | 12 | 暗い緑 |
| 1 | 黒 | 5 | 明るい青 | 9 | 明るい赤 | 13 | 紫 |
| 2 | 緑 | 6 | 暗い赤 | 10 | 黄 | 14 | 灰 |
| 3 | 明るい緑 | 7 | 水色 | 11 | 明るい黄色 | 15 | 白 |

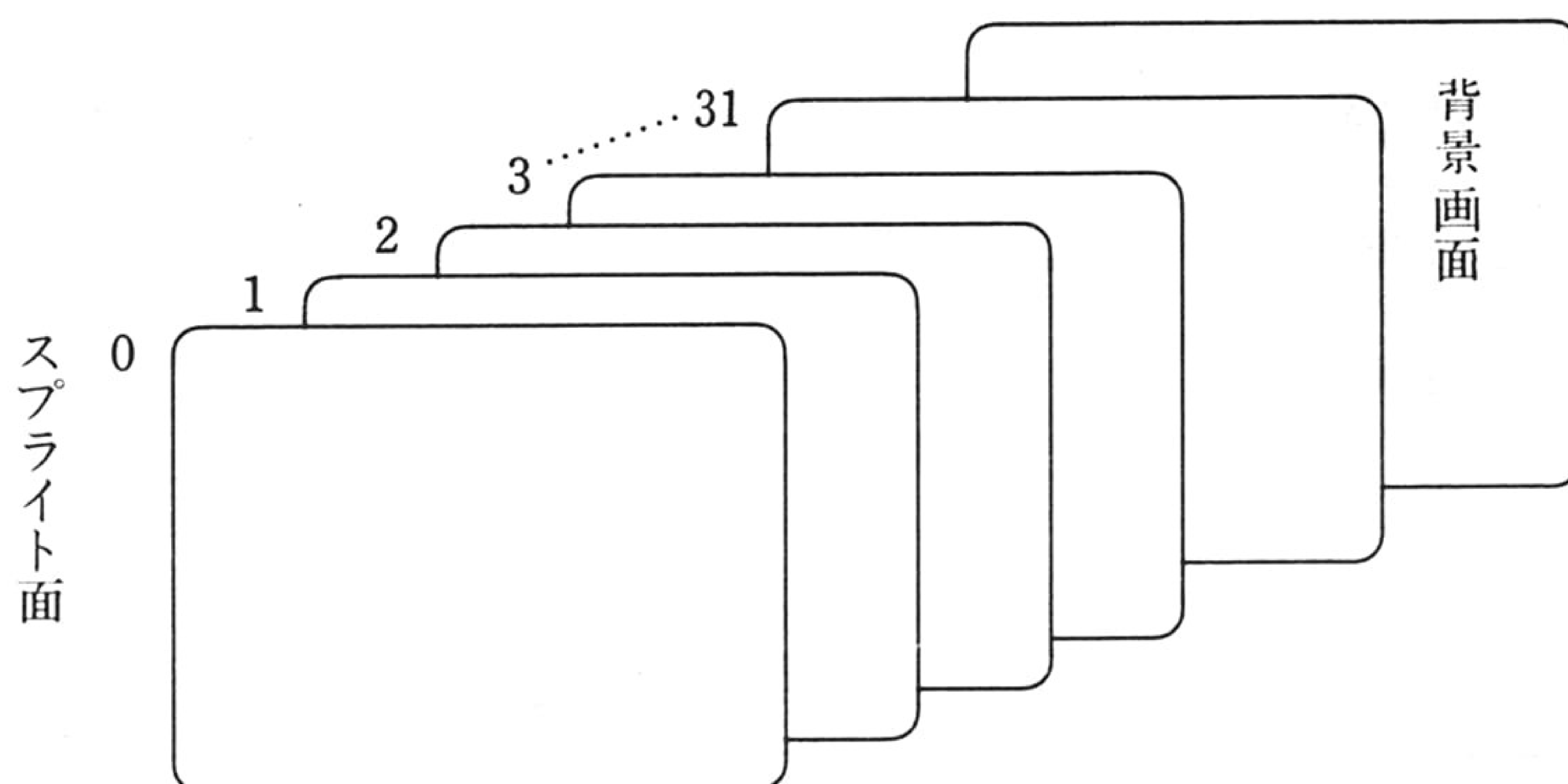
## 1-17 サウンド

8オクターブの音域と3重和音をサポートするPLAY文、およびサウンドジェネレータのレジスタを直接変更するSOUND文が用意されています。

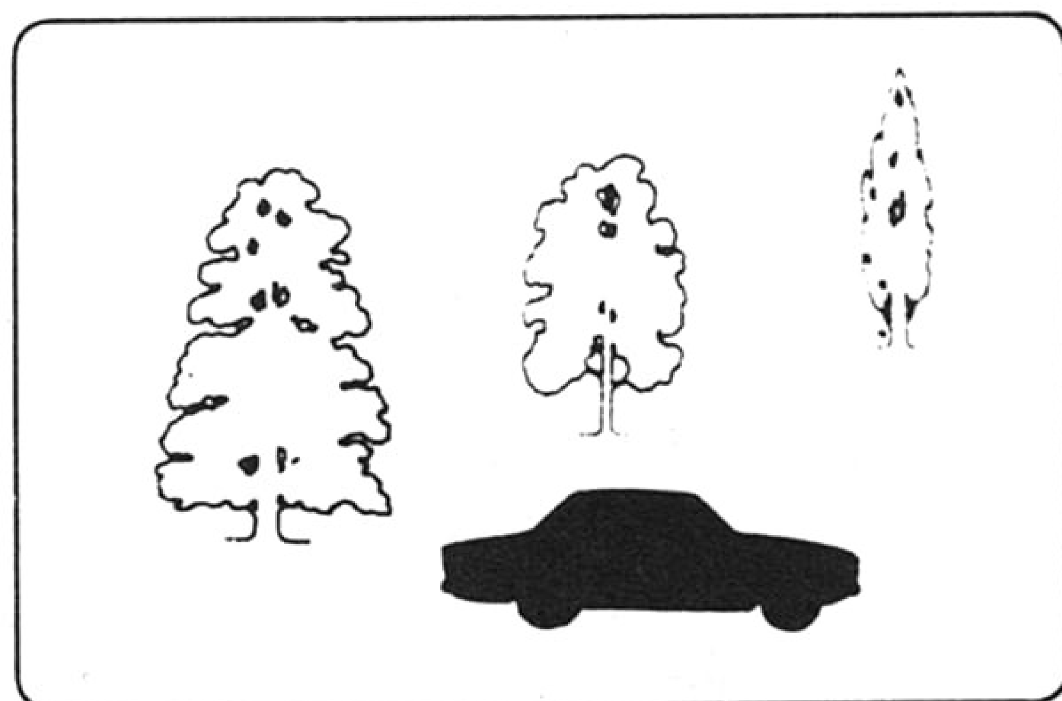
- PLAY文
  文字列を与えることで音楽を演奏することができます。
- SOUND文
  SOUND文を使うと、効果音や疑似音を出すことができます。

## 1-18 スプライト

MSX BASICでは、スプライト（動画）機能があります。これは、グラフィック画面やテキスト画面とはまったく別に、スプライト面というのがあるのです。スプライト面とは、画面の前に重ねてあるもうひとつの面と考えられるものです。そして、これには0から31番まで32枚が用意されており、プログラムで定義したスプライトパターンを1つずつ表示できます。




1つのスプライトの大きさは8×8ドット、または16×16ドットですが、それぞれ2倍に拡大して表示することも可能です。
スプライトパターンは8×8ドットの場合256個、16×16ドットの場合64個まで定義することができます。
なお、スプライト機能とは、スプライト面にスプライトパターンを表示すると、画面にあった絵や文字の上にスプライトパターンが重なったようにみえ、移動しても絵や文字の表示に影響を与えないことです。



※車を動かしても、うしろの木の絵はそのままのこり、動画になります。

スプライトパターンは同時に32個まで表示できますが、水平方向に5つ以上並ぶと、前面から5つ目以降のスプライトは表示されません。

# 第2章
# コマンド解説

## 第2章の見方

■機能　命令の機能を説明しています。

■書式　命令の書き方を示します。これに従って入力してください。

- 命令は大文字でも小文字でもかまいません。ただし、引用符（"）で囲まれた文字列は、大文字と小文字を区別する必要があります。
- カギカッコ（{ }）で囲まれた項目はユーザーが指定します。
- 角カッコ（[ ]）で囲まれた項目はオプションで省略することができますが、省略された場合、省略値または以前に指定した値が適用されます。
- カンマ（,）で区切られる複数のパラメータ（引数(ひきすう)）があって、省略可能な場合、あとにあるものを全部省略する場合は、カンマも省略しますが、前の方を省略してあとのものを指定するときは、間にあるカンマは省略できません。
- パラメータとは命令を実行する上で必要とされる値です。たとえば、表示桁数を変更するWIDTH文で何桁の表示を行なうかを指定しなければなりません。WIDTH 20とすれば、最大20桁になります。このときWIDTH文に対するパラメータは20となります。

〔例〕 SCREEN 2　　　（2つ目以降のパラメータを省略した）
　　　SCREEN ,,,1　（前の3つのパラメータを省略した）

■文例　実際の入力の見本として、簡単な例が示されています。

■範囲　引数などがとれる範囲が示されています。

■解説　命令の正しい使用法、詳しい機能、注意点が説明されます。
文中の ⌐ は、例えば CTRL ⌐ STOP のときは、CTRL キーを押しながら STOP キーを押すという意味です。

■サンプルプログラム；
命令を使ったサンプルプログラムです。実際に入力して確かめてください。またプログラム中のREMは、アポストロフィー（'）で略しています。

注）本書では、プログラムリストを見やすくするために、プリンタ印字したリストを使用してあります。
この場合、アスタリスクマーク（*）が（*）となっていますので入力の際はご注意ください。

# ☞INDEX

# AUTO 〈コマンド〉
オート

**機能** 行番号を自動的に発生させます。

**書式** AUTO〔{行番号}〕〔,{増分}〕

**文例** AUTO　100, 10

**範囲** 行番号の許される範囲は０～65529の整数

**解説** AUTOコマンドはプログラムの入力が楽になるように、指定された{行番号}を、画面に表示し、行の入力待ちにします。行の入力が終わる(リターンキーが押される)ごとに{増分}ずつ増加した行番号が自動的に発生します。

{行番号}だけを省略すると０が省略値になります。

{増分}を省略すると10が省略値になります。

CTRL → C または CTRL → STOP を押すと、AUTOコマンドから抜けることができます。このとき、最後に表示された行の入力は無効となります。

プログラム中に、すでにある行番号が発生されたときには、行番号に直接アスタリスク(*)が表示されます。このとき文字を入力し、リターンキーを押すと、入力した文字に入れかわりますが、ただ単にリターンキーを押せば、その行の内容は以前のままになります。

AUTOモード中に、スクリーンエディット機能を利用して、カーソルをすでに入力した行、あるいは、表示されている行へ移動し、リターンキーを押した場合、その行番号に増分を加えた行番号が新たに発生されます。

# RENUM 〈コマンド〉
リナンバー

**機能** プログラムの行番号のつけ直しを行ないます。

**書式** RENUM〔{新行番号}〕〔,{旧行番号}〕〔,{増分}〕

**文例** RENUM 1000

**範囲** 行番号は０～65529の整数

**解説** プログラムの行番号を自由につけ直します。

{新行番号}は、新しくつける開始行番号です。

{旧行番号}は、プログラムのつけ直しをはじめる行番号です。これを省略すると最初の行からつけ直しがはじまります。

{増分}は、プログラムの行番号の間隔です。省略値は10です。

この命令は、GOTO、GOSUB、IF～THEN、ON～GOTO、ON～GOSUB、ERL 文などで参照している行番号も処理の対象となります。これらの文が参照している行番号がないときは“Undefined line xxxxx in yyyyy”というエラーメッセージになります。xxxxx は実際に存在しなかった行番号で、RENUM を実行しても変更されません。yyyyy はつけかえる前の行番号になります。

RENUM コマンドを実行して、行番号が 65529 を越えるときは、“Illegal fanction call”エラーになります。

# DELETE 〈コマンド〉

## デリート

**機能** プログラムを部分的に削除します。

**書式**

DELETE | {始点行番号} − {終点行番号} / {行番号} / − {終点行番号} |

**文例** (1)DELETE 50-100 (2)DELETE 50 (3)DELETE -300

**範囲** 行番号は 0 ～65529の整数

**解説** {始点行番号} から {終点行番号} までのプログラムを削除します。ただし、{行番号} だけを指定した場合は、その行だけ削除します。-{終点行番号} を指定した場合には、プログラムの先頭から {終点行番号} までを削除します。

行番号の指定ではBASICが現在着目している行の行番号の代わりにピリオド(.)を使うことができます。たとえば、プログラム実行中にエラーが発生した行を消すときは、エラーでプログラムが中断したあとにDELETE . とします。なお、{始点行番号} を省略するとプログラムの先頭からになりますが、{終点行番号} を省略することはできません。

行番号の存在しない行番号を指定すると "Illegal function call" エラーとなります。

**サンプルプログラム：**

```
10 A=10
20 B=30
30 C=A+B
40 D=A-B
50 E=A*B
60 F=A/B
70 PRINT "A=";A
80 PRINT "B=";B
90 PRINT "A+B";C
100 PRINT "A-B";D
110 PRINT "A*B";E
120 PRINT "A/B";F
DELETE 70-80 [↵]
LIST [↵]
10 A=10
20 B=30
30 C=A+B
40 D=A-B
50 E=A*B
60 F=A/B
90 PRINT "A+B";C
100 PRINT "A-B";D
110 PRINT "A*B";E
120 PRINT "A/B";F
```

70行～80行のプログラムをDELETE文で削除してみました。

# NEW
〈コマンド〉

ニュー

**機能** メモリーにあるプログラムを削除し、すべての変数をクリアーします。

**書式** NEW

**文例** NEW

**解説** すべての変数をクリアー(初期状態にする)し、開かれているファイルがあれば、それも自動的に閉じます。また、DEF FN 文による定義も解消されます。
プログラムを新しく入力するときに使用します。

**参照** DELETE, ERASE

**サンプルプログラム：**

```
10 PRINT "  ﾌﾟﾛｸﾞﾗﾑ ﾁｭｳ ﾆ NEW ｦ ｲﾚﾙﾄ"
20 PRINT "  ﾌﾟﾛｸﾞﾗﾑ ﾊ ｷｴﾃｼﾏｲﾏｽ"
30 PLAY "O4GO5C"
40 FOR J=1 TO 2000 : NEXT J
50 PRINT TAB(10);"NEW"
60 NEW
```

プログラムの実行が終了するとプログラムはなくなっています。

# LIST/LLIST
〈コマンド〉

リスト/エルリスト

**機能** メモリーにあるプログラムの全部または一部を画面に表示したり、プリンタに出力したりします。

**書式** (1) LIST〔{行番号}〕〔−〔{行番号}〕〕
(2) LLIST〔{行番号}〕〔−〔{行番号}〕〕

**文例** (1) LIST 100 − 200
(2) LLIST 300 −

**範囲** 行番号は0～65529の整数

**解説** プログラムのリストを画面に表示するのが LIST で、プリンタに打ち出すのが LLIST です。
すべての行を表示または印字したいときは{行番号}を省略します。
指定した行番号だけのときは、{行番号}を指定します。
表示または、印字の範囲を決めるときは、次のようになります。

LIST ↵ (先頭行から表示)
LIST20 ↵ (行番号20を表示)
LIST50−100 ↵ (行番号50から行番号100までを表示)
LIST200− ↵ (行番号200以降を表示)
LIST−80 ↵ (先頭行から行番号80までを表示)

{行番号}の代わりにピリオド(.)を指定すると、BASIC が着目している行になります。

LIST, LLISTコマンドの中断は、CTRL → STOP キーを押します。

# CLOAD/CLOAD?

〈コマンド〉

**シーロード/シーロードクエスチョン**

**機能** プログラムをカセットテープからメモリーへ読み込みます。

**書式** CLOAD〔"{ファイル名}"〕
CLOAD ?〔"{ファイル名}"〕

**文例** CLOAD "SAMPLE"
CLOAD ? "SAMPLE"

**範囲** ファイル名の長さは6文字までで、7文字以上は先頭の6文字まで有効。

**解説** カセットテープにCSAVEコマンドでセーブされたBASICのプログラムをロード(メモリー上に格納) します。
ファイル名を省略すると、最初に見つけたプログラムをロードします。
指定したファイル名を見つけたときは "Found：ファイル名" と表示し、ロードされますが、指定したファイル名と異なるときは"Skip：ファイル名"と表示されてロードされません。この場合は、"Found"が表示されるまでテープを読み込みます。
テープの読み込みのエラーが起きたり、途中で CTRL → STOP キーを押したりしたときは "Device I/O error" が表示されます。読み込み中にエラーが起きたり、いつまでも"Found"が表示されないときは、テープレコーダーの音量を変えて再びロードしてください。
CLOAD ? は、カセットテープ上のプログラムとメモリー上のプログラムが同一か異なるかをチェックします。同一であれば "OK" 、異なるプログラムならば、"Verify error"と表示されます。なお、同一プログラムであっても、メモリー容量の異なる状態でCSAVEされたプログラムをチェックしたときや、読み込みエラーが発生したときは"Verify error"が表示されることもあります。
ロードするプログラムが、BASICのフリーエリアよりも大きい (メモリーが少ないときや CLEAR 文の値が適切でない)ときは"Out of memory"エラーが表示されます。
CLOADを実行し、"Found"が表示されると今までのプログラムは消され、変数はクリアーされ、ファイルも CLOSE されます。ボーレートは自動的に決定されるので、指定する必要はありません。

**参照** CSAVE

## CSAVE 〈コマンド〉
シーセーブ

**機能** メモリーにあるプログラムをカセットテープに書き込みます。

**書式** CSAVE〔{"ファイル名"}〕〔{,ボーレート}〕

**文例** CSAVE "TEST"

**範囲** ファイル名は6文字までで、7文字以上を指定しても、7文字以降は無視します。

**解説** メモリー上のプログラムをバイナリ形式でカセットに記録します。この記録（セーブ）されたプログラムはCLOADで読む(ロードする)ことができます。
{ボーレート}は1と2を指定できます。
1：1200ボー
2：2400ボー
省略するとSCREEN文で指定された値になります。指定のないときは1200ボーです。CTRL → STOPでCSAVEの実行を中断できます。

**参照** CLOAD, BSAVE, SAVE

## SAVE 〈コマンド〉
セーブ

**機能** メモリーのプログラムをアスキー形式でファイルに読み込みます。

**書式** SAVE {"ファイル名"}

**文例** SAVE "CAS：CASIO"

**範囲** {ファイル名}は最初の6文字までが有効。

**解説** {ファイル名}で指定したファイルにアスキー形式でプログラムをセーブします。
CSAVEでセーブされたプログラムはCLOADで読むこと（ロード）ができますが、SAVEでセーブしたプログラムはLOADで読み込みます。
{ファイル名}はカセットレコーダーを使う場合は"CAS："を指定します。
アスキー形式とは、プログラムを構成する1文字1文字をアスキーモードとしてセーブする方式です。これに対し、CSAVEではバイナリ形式と呼ばれる方式でコマンドや関数を中間言語におき換えて圧縮してセーブします。
アスキー形式は、バイナリ形式より多くのバイト数を必要としますが、MERGEコマンドを使うことができます。またデータファイルとして、LINE INPUT で行単位をひとつのデータとして取り扱うことができます。
各行の区切りは、キャリッジリターンコード（CHR$(13))、ラインフィードコード(CHR$(10))で区切られ、最後はCTRL → Zコード(CHR$(26))が付け加えられます。
カセットテープに記録するときのボーレートの指定はSCREEN文で行ないます。

**参照** LOAD, MERGE, CSAVE, SCREEN

## LOAD　〈コマンド〉
ロード

**機能**　アスキーファイルとして作成されたプログラムをメモリーに読み込みます。

**書式**　LOAD "{ファイル名}"〔, R〕

**文例**　LOAD "CAS : CASIO"

**解説**　{ファイル名} で指定したファイルをロードします。このファイルは必ず SAVEコマンドでセーブされたファイルでなければなりません。
カセットテープからのロードはファイル名中のデバイス名に"CAS :"を指定します。
,R オプションをつけると、ロード後すぐに実行します。
カセットテープでファイル名を省略すると、最初に見つけたファイルをロードすることになっています。
LOAD 命令ではファイルはクローズされますが、, R オプションをつけると、オープンしているファイルはクローズされません。

**参照**　SAVE, MERGE, CLOAD

## BSAVE　〈コマンド〉
ビーセーブ

**機能**　メモリーにある機械語プログラムをファイルに書き込みます。

**書式**　BSAVE"{ファイル名}", {開始アドレス}, {終了アドレス}〔, {実行開始アドレス}〕

**文例**　BSAVE"CAS : SAMPLE", &HA100, &HA2FF

**範囲**　開始アドレス、終了アドレス、実行アドレスは0 ~ 65535

**解説**　{開始アドレス} から {終了アドレス} までの範囲のメモリーの内容を {ファイル名} で指定したファイルにセーブします。
{ファイル名} は、カセットレコーダーを使う場合は"CAS :"を指定します。
また、{実行開始アドレス} を指定すると、BLOAD コマンドでRオプションを付けたときの機械語プログラムを自動的に実行することができます。省略すると、開始アドレスが実行開始アドレスとみなされます。

**参照**　BLOAD

# BLOAD 〈コマンド〉
## ビーロード

**機能** 機械語プログラムをメモリーに読み込みます。

**書式** BLOAD "{ファイル名}"〔,R〕〔,{オフセット}〕

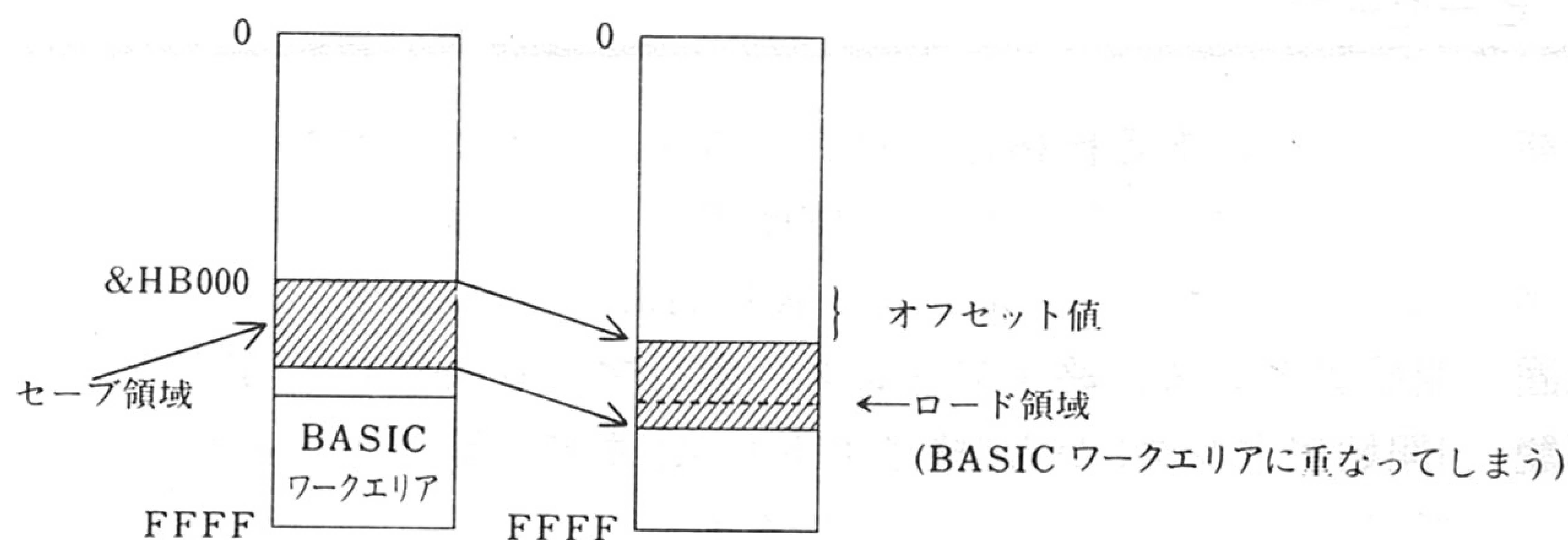
**文例** BLOAD "CAS：SAMPLE", R (CAS：カセットを意味します。)

**範囲** {オフセット}は0～65535

**解説** {ファイル名}で指定した機械語、プログラムファイルをメモリー上にロードします。ファイル名を省略すると、カセットテープに記録された最初の機械語をロードします。{オフセット}を指定すると、セーブ時に指定された開始番地に{オフセット}を加えた値をロードする番地としてロードします。したがって{オフセット}をつけてロードするプログラムはセーブされたときと異なる番地になっても実行可能でなければなりません。

(注) メモリーのどの部分にも読み込めるので、オフセットの値などによっては、BASICで使用しているワークエリアやファイルコントロールブロックの部分に読み込んでしまい、暴走してしまう場合があるので注意が必要です。

Rオプションを指定すると、プログラムをロードしたあとBSAVEで指定しておいた実行開始アドレスから実行されます。このとき、すでにファイルが開かれている場合、そのファイルは開かれたまま機械語プログラムが実行されています。なお、オフセットを省略するときには、カンマ(,)を含めて省略します。また、ロード中にエラーが起きたときは"Device I/O error"が表示されます。




**参照** BSAVE

# MERGE　〈コマンド〉

マージ

**機能**　アスキーファイル上のプログラムをメモリーのプログラムに混合し、ひとつのプログラムにする。

**書式**　MERGE "{ファイル名}"

**文例**　MERGE "CAS：CASIO"

**解説**　メモリー上のプログラムに{ファイル名}で指定したプログラムファイルを混合してひとつのプログラムにし、メモリー上におきます。
{ファイル名}で指定するファイルはSAVE文を使ってアスキー形式でセーブされていなければなりません。
プログラム中でこのコマンドを実行すると、実行後コマンドレベルに戻ります。
{ファイル名}はカセットレコーダーを使う場合のデバイス名には"CAS："を指定します。また、{ファイル名}のファイル名を省略すると、カセットテープの位置している場所以後にある最初のアスキーファイルがロードされます。
ファイル中のプログラムと、メモリー中のプログラムに同一の行番号があった場合には、ファイル中の行でメモリー中の行をおき換えます。

**参照**　SAVE, LOAD

# CLEAR　〈ステートメント〉

クリアー



**機能**　変数を初期化し、また、メモリー領域の大きさを設定します。

**書式**　CLEAR〔[{文字列領域の大きさ}],〔{メモリー上限}]〕

**文例**　CLEAR 400

**範囲**　文字列領域はメモリーの許される限り、メモリーの上限は16KRAMで&HC31F ~ &HF380，32KRAMで&H831F ~ &HF380

**解説**　すべての数値変数を0，文字変数を" "(ヌルストリング) に初期化します。また、DEF文で定義した情報もすべて無効になります。ファイルもすべてCLOSEします。

{文字列領域の大きさ}は文字変数の内容を記憶する文字列領域の大きさとなります。

{メモリーの上限}は、指定された番地の直前番地までをBASICで使うことができます。その番地から&HF830番地までの値はBASICで変化することはありませんので、機械語プログラム等を置くことができます。

値が適切でないときは、"Illegal function call"エラーになります。

メモリー上限を指定するときは{文字列領域の大きさ}のみを省略することはできません。

**サンプルプログラム：**

```
10 A=2:D$="abcdef"
20 PRINT USING"Clear ﾏｴ A=### ";A;
30 PRINT " D$=";CHR$(34)+D$+CHR$(34)
40 PRINT
50 CLEAR
60 PRINT USING"Clear ｱﾄ A=### ";A;
70 PRINT " D$=";CHR$(34)+D$+CHR$(34)
80 END
```

文字変数、数値変数を使用したときの変数の値が、50行のCLEAR文によって変更される様子をみるプログラムです。

# FRE 〈関数〉

フリー

**機能** メモリーの未使用領域の大きさを与えます。

**書式** FRE（{引数}）

**文例** A＝FRE(" ") その結果 A＝(未使用の文字領域の大きさ）となる。

B＝FRE（0）その結果 B＝(未使用のプログラムおよび変数領域の大きさ）となる。

**範囲** {引数} は数値変数、数値定数、文字変数、文字定数いずれでもかまいません。

**解説** FRE関数は使用可能なメモリー領域のうちの未使用領域の大きさを与えます。

FRE（{数値}）はテキスト(プログラム)領域、変数領域などのユーザーエリアの未使用領域の大きさを与えます。

FRE({文字列})は、文字列領域の未使用エリアの大きさを与えます。また、文字列領域中の不用となった文字列を削除して、使用可能領域を広げます。このことをガベージコレクションと呼びます。

**参照** 付録「メモリーマップ」, CLEAR

**サンプルプログラム：**

```
10 PRINT FRE(0);FRE("")
20 DIM A%(100)
30 PRINT FRE(0);FRE("")
40 FOR J=0 TO 20
50   A$=A$+"a"
60 NEXT J
70 PRINT FRE(0);FRE("")
80 END
```

プログラム実行前のフリーエリアの大きさと、文字列領域の大きさを表示します。その後、数値型配列を定義したあとの領域の大きさ、文字変数を使ったあとの領域の大きさを表示しています。

## SCREEN 〈ステートメント〉
スクリーン



**機能** 画面のモード、スプライトサイズ、キークリックスイッチ、カセットボーレート、プリンタオプションを指定します。

**書式** SCREEN〔{画面モード}〕〔,{スプライトサイズ}〕〔,{キークリックスイッチ}〕〔,{カセットボーレート}〕〔,{プリンタオプション}〕

**文例** SCREEN 2, 2

**範囲** {画面モード}は0～3、{スプライトサイズ}は0～3、{キークリックスイッチ}は0～1、{カセットボーレート}は1～2、{プリンタオプション}は0または0以外。

**解説** MSXの画面モードを切り換えると同時に、プリンタやカセットに対しての設定を行ないます。

{画面モード}は次のようになります。

0：最大40×24文字で構成されるテキストモード
1：最大32×24文字で構成されるテキストモード
2：256×192画素(ドット)で表わすことのできる高解像度グラフィックモード
3：64×48ブロックで構成されるマルチカラーモード。1ブロックの大きさは高解像度モードと較べて、縦横それぞれ4倍になります。

画面モードを切り換えると、今まで表示されていた画面は消えます。

テキストモードでは、リストを表示することができますが、グラフィック画面ではそれができません。また、画面モード0ではスプライトの表示はできませんし、コマンドレベルに戻るとテキストモードになってしまいます。

{スプライトサイズ}は{式}の値により、次のように定義されます。

0：スプライトを8×8ドットで構成します。
1：スプライトを8×8ドットで構成し、それを縦横2倍に拡大します。
2：スプライトを16×16ドットで構成します。
3：スプライトを16×16ドットで構成し、それを縦横2倍に拡大します。

ここで指定された値は、SPRITE$でスプライトパターンを定義するときのドットの構成になります。

{キークリックスイッチ}は、キーを押すごとにクリック音（キーを押したときに出る音）を出すかどうかの設定をします。

省略時または0：音を出さない。
1：音を出す。

{カセットボーレート}は、カセットテープに書き出すときのボーレートを指定します。CSAVE, BSAVE, SAVE, PRINT#コマンドでボーレートを省略したとき有効になります。

CLOAD, BLOADでは自動的にセーブされたボーレートの値をとります。

省略時または1：1200ボー
2：2400ボー(1200ボーの2倍の速度になります。)

{プリンタオプション}は、使用しているプリンタの文字構成がグラフィック文字とひらがなを持っているかどうかを指定します。MSX標準プリンタを選択すれば、グ

ラフィック文字とひらがな文字をそのまま出力しますが、MSX 標準以外のプリンタに設定すれば、グラフィック文字は空白に、ひらがな文字はカタカナにおきかえて出力されます。

それぞれの値が適切でないときは“Illegal function call”エラーになります。

**サンプルプログラム：**

```
10 SCREEN 0:KEY OFF:WIDTH 40:X=40*23 :PRINT TAB(10);"SCREEN 0"
20 GOSUB 130
30 SCREEN 1:WIDTH 32:X=32*23:PRINT TAB(6);"SCREEN 1"
40 GOSUB 130
50 SCREEN 2:PSET(0,0)
60 OPEN "GRP:" FOR OUTPUT AS #1
70 PRINT #1,"     SCREEN 2"
80 FOR J=1 TO 32*23
90   PRINT #1,CHR$(64+INT(RND(1)*27));
100 NEXT J
110 FOR J=1 TO 2000:NEXT J
120 END
130 FOR J=1 TO X
140   PRINT CHR$(64+INT(RND(1)*27));
150 NEXT J
160 RETURN
```

SCREEN 0 ～ 2 までの表示のちがいを示すプログラムです。

# KEY 〈コマンド〉

## キー

**機能**　キーボードの上部にあるファンクションキーの内容を定義します。

**書式**　KEY {キー番号}，{文字列}

**文例**　KEY 2, "AUTO"（ファンクションキー2番 F2 の内容を AUTO に定義します。）
KEY 3, "CLS : RUN" + CHR$(13)（ファンクションキー3番 F3 の内容を CLS : RUN ↵ に定義します。）

**範囲**　{キー番号}は1～10、{文字列}は最大15文字まで。

**解説**　{キー番号}で指定するファンクションキーの内容を{文字列}で定義します。
{キー番号}は1～10で、それぞれ F1 ～ F10 に対応しています。
{文字列}は最大15文字までの文字およびコントロール文字です。キーボードから入力できない文字は、CHR$ 関数を使います。
本体の電源ON時は、F1～F5にあらかじめセットされている内容が画面下に表示されています。このとき SHIFT を押すと、F6～F10の内容が表示されます。

**サンプルプログラム**：

```
10 FOR J=1 TO 10
20  KEY J,CHR$(64+J)+CHR$(13)
30 NEXT J
40 KEY LIST
50 END
```

ファンクションキーの定義をするプログラムです。

# KEY LIST 〈コマンド〉

## キーリスト

**機能**　ファンクションキーの内容を画面に表示します。

**書式**　KEY LIST

**文例**　KEY LIST

**解説**　ファンクションキーの内容の一部はテキスト画面の最終行に表示させておくことができますが、KEY LISTで F1 ～ F10 までのすべての内容を画面に表示します。

# KEY ON/OFF 〈ステートメント〉

キー オン/オフ

**機能** ファンクションキーの内容をテキストモードの画面の最下行に表示するか、しないかを指定します。

**書式** KEY ON
KEY OFF

**文例** KEY ON
KEY OFF

**解説** ファンクションキーの表示を行なったり、やめたりします。
KEY ON がファンクションキーの表示で、KEY OFF がとりやめになります。
KEY ON の場合、通常はファンクションキー1～5が表示され SHIFT キーが押されている間はファンクションキー6～10が表示されます。

**参照** KEY

**サンプルプログラム：**

```
10 KEY ON
20 IN INKEY$="" THEN 20
30 KEY OFF
40 IF INKEY$="" THEN 40
50 GOTO 10
```

何かキーが押されるごとにKEY ONとKEY OFFが実行されます。

# RUN 〈コマンド〉

ラン

**機能** プログラムの実行を開始します。

**書式** RUN〔{行番号}〕

**文例** RUN 200

**範囲** {行番号}は、0～65529

**解説** プログラムの実行を行ないます。行番号を省略すると、プログラムの先頭から実行されます。
すべての変数は初期化され、ファイルはクローズされてからプログラムが実行されます。

# CONT 〈コマンド〉

コンティニュー

**機能** CTRL + STOP キー入力、または STOP 文によって停止したプログラムの実行を再開します。

**書式** CONT

**文例** CONT

**解説** プログラムの実行を中断したときに、実行の再開を行ないます。中断とは、CTRL + STOP キーが押されたとき、または STOP 命令を実行したときです。

プログラムを中断したあとにプログラムの追加、訂正、削除などをすると、再開できなくなります。このときは“Can't CONTINUE”と表示されます。LET 文で変数の値を変更したり PRINT 文をダイレクトモードで実行して、変数の値を調べたりした後の再開はできるので、プログラムのデバックの際に有効です。

CONT はプログラム中には入れないでください。入れるとやはり“Can't CONTINUE”が表示されます。

INPUT 文で入力待ちの状態にあるときに停止した場合は、その INPUT 文から再開されます。

**サンプルプログラム：**

```
10 PRINT "start"
20 J=1
30 PRINT J:STOP
40 J=J+1:GOTO 30
```

30行のSTOP文でプログラムはそのつど中断しますので、CONT ↵ で再開します。Jの変化するようすがよくわかると思います。

# TRON/TROFF 〈コマンド〉

トレースオン／トレースオフ

**機能** プログラムの実行状態を調べます。

**書式** TRON
TROFF

**文例** TRON
TROFF

**解説** プログラムの実行をトレースしたり、トレースを解除したりします。

トレースとは、プログラムがどのように実行されるかを確認するためのもので、実行したプログラムの行番号を角カッコ〔 〕で囲んで表示します。

TRONでトレースモードになり、TROFFまたはNEWコマンドで解除されます。

TRON文の実行はダイレクトモードでもプログラムモードでも可能です。

**サンプルプログラム：**

```
10 TRON
20 FOR J=1 TO 10
30 NEXT J
40 TROFF
50 END
```

TRON文を使って、画面に行番号が表示される状態を観察してみます。

# MAXFILES 〈ステートメント〉

マックスファイルズ

**機能** 使用するファイルの数を定義します。

**書式** MAXFILES ＝{ファイル数}

**文例** MAXFILES ＝ 4

**範囲** ファイル数は 0 ～15

**解説** OPEN 文で開いて使用するファイルの最大数を定義します。以後は、指定した数のファイルを同時に開いて使用することができます。

ファイル 1 個について 267 バイトのファイルコントロールブロック領域が確保されますので、必要なファイル数のみを指定しないとメモリー不足につながります。

オープンできるファイル数の初期値は 1 個です。

MAXFILES 文を実行すると、変数がすべて初期化され、現在オープンしているファイルはクローズされます。

**サンプルプログラム：**

```
10 MAXFILES=1
20 PRINT "MAXFILES= 1:FRE=";FRE(1)
30 MAXFILE=15
40 PRINT "MAXFILES=15:FRE=";FRE(1)
50 END
```

MAXFILESの値によってメモリーの空き容量を比較するプログラムです。

# DEF INT/SNG/DBL/STR〈ステートメント〉

ディファインイント/シングル/ダブル/ストリング

**機能** 変数の型を宣言します。

**書式**

DEF {INT / SNG / DBL / STR} {文字の範囲}〔,{文字の範囲}……〕

**文例** DEFINT A−Z

DEFINT A, X−Z 先頭がA, X, Y, Z, の文字で始まる変数が整数型となります。

**範囲** 英文字を1文字、または(英文字1文字)〜(英文字1文字)

**解説** DEF文を実行すると、{文字の範囲}で指定された文字で始まる変数名の型を宣言します。

DEFINT——整数型
DEFSNG——単精度実数型
DEFDBL——倍精度実数型
DEFSTR——文字型

DEF文で型宣言を行なっても、型宣言文字(%、!、#、$)による指定が優先されます。具体的に例を示すと、BASICでDEFINT A−Zとした後、A=7/5にすればAは整数型となり結果は1になりますが、A!=7/5とすれば、A!は単精度となり結果は1.4となります。型宣言をDEF文で実行しても、その後に変数名の最後に%、!、#、$を付加することにより自由に型を使用できるわけです。型宣言を行わないとき、型宣言文字をつけない変数は全て倍精度変数になります。CLEAR文を実行するとDEF文の宣言は無効になります。

**サンプルプログラム**：

```
10 A!=1/3:PRINT A!
20 DEFINT A
30 A=4/3:PRINT A
40 DEFDBL A
50 A=1/7:PRINT A
60 DEFSNG A
70 PRINT A
80 DEFSTR A
90 A="abcdefg"
100 PRINT A
110 END
```

Aという変数をそれぞれ整数型、単精度型、倍精度型、文字型に定義して計算結果を表示します。

# DEF FN

〈ステートメント〉

ディファインファンクション

**機能** ユーザーによって作られた関数を定義します。

**書式** DEF FN {名前} 〔({引数})〕＝{関数の定義式}

**文例** DEF FNP (X, Y) ＝X＊4＋Y＊3

**範囲** {名前}は、変数名の規則にもとづいた1～2文字の英数字です。

**解説** 関数を自分で定義するときに使います。この定義された関数はプログラム中で、何回も利用できます。

{引数}は、{関数の定義式}中で使われる同じ名前の変数に対応されます。この変数はプログラム中にある変数とは同一変数名でもまったく関連をもちません。

{関数の定義式}の中で使われている変数を{引数}で指定していない場合は、その変数が関数を実行した時点で持っている値が使われます。

{関数の定義式}の範囲は1行です。引数の型は自由に使うことができますが、DEF FN文で定義した引数は、変数の並びと型が実行するときの並びと型に合っていなければなりません。この文で定義される前に関数を実行することはできません。

また、DEF FN文の指定にまちがいがあると、関数を実行するときに"Syntax error"が表示されます。

プログラム中で定義した関数を使用するときは, FN {名前} 関数として使用できますが、使用する前にDEF FN文で関数が定義されなければなりません。

**サンプルプログラム：**

```
10 PI#=3.1415926536#
20 DEF FNSN(X)=SIN(X/180*PI#)
30 DEF FNCS(X)=COS(X/180*PI#)
40 SCREEN 2
50 LINE(0,96)-(255,96)
60 LINE(0,0)-(0,191)
70 FOR J=0 TO 360
80   X=J/360*255
90   Y1=96-63*FNSN(J)
100  Y2=96-63*FNCS(J)
110  PSET(X,Y1):PSET(X,Y2)
120 NEXT J
130 IF INKEY$<>" "THEN 130
140 END
```

SIN, COSの関数は、ラジアンで表現されますが、度で計算する関数をDEF FN文で定義しています。
プログラムを実行すると0～360°のSIN, COSのグラフを表示します。スペースキーを押せば終了します。

定義宣言

# DEF USR 〈ステートメント〉

ディファインユーザー

**機能**　機械語で作られたユーザー関数の実行開始番地を定義します。

**書式**　DEF USR〔{番号}〕={開始番地}

**文例**　DEF USR 5 =&HE000

**範囲**　番号は0〜9

**解説**　USR関数が呼び出す機械語ルーチンの実行開始番地を定義します。複数のUSR関数を用いる場合の識別を行ないますので、最大10個まで使うことができます。{番号}が省略されたときは0とみなされます。

なお、{開始番地}は機械語ルーチンの実行開始番地です。

この文は二重に定義してもエラーにならず、後から定義したものが有効になります。

**参照**　USR，CLEAR

**サンプルプログラム：**

```
10 CLEAR 200,&HE000
20 DEF USR=&HE000
30 FOR J=0 TO 23
40 READ D$
50 POKE &HE000+J,VAL("&H"+D$)
60 NEXT J
70 A=USR(0)
80 END
90 DATA 21,15,E0,7E
100 DATA 32,E9,F3,23
110 DATA 7E,32,EA,F3
120 DATA 23,7E,32,EB
130 DATA F3,CD,62,00
140 DATA C9,06,0F,07
```

機械語を使って画面の色を変化させるプログラムです。機械語の入力をまちがえると暴走してしまいます。

定義宣言

# DIM　〈ステートメント〉

## ディメンジョン

**機能**　配列変数を定義し、メモリー領域を割り当てます。

**書式**　DIM {変数名} ({添字の最大値}〔,{添字の最大値}……〕)

**文例**　DIM X(10,15)

**範囲**　添字の最大値はメモリーの許すかぎり。

**解説**　配列変数の添字の最大値を設定します。DIM文を実行したとき、数値変数は、要素の値が0となり、文字型の場合はヌルストリング(" ")になります。
DIM文で宣言しない場合は、配列の要素は最大値10です。
また、不用になった配列変数はERASE文で削除することができます。設定された配列の最大値より大きい添字を使うと"Subscript out of range"エラーが発生し、一度定義した配列名と同じ配列名を定義してしまったときは、"Redimensioned array"エラーになります。メモリー領域が不足している場合は"Overflow"エラーになります。

**参照**　ERASE

**サンプルプログラム**：

```
10 FOR J=0 TO 10
20 A(J)=RND(1)
30 NEXT J
40 FOR J=0 TO 10
50 PRINT J;A(J)
60 NEXT J
70 IF INKEY$<>" " THEN 70
80 ERASE A:PRINT
90 DIM A(20)
100 FOR J=0 TO 20
110 A(J)=RND(1)
120 NEXT J
130 FOR J=0 TO 20
140 PRINT J;A(J)
150 NEXT J
160 END
```

10行～60行ではDIM文を使わないで、配列を使っています。90行～150行では、配列の大きさを20で使うため、80行で、今まで使った配列を削除したあとに、DIM文を実行しています。

定義宣言

## ERASE　〈ステートメント〉
イレーズ

**機能**　配列変数を消去します。

**書式**　ERASE{配列変数名}〔,{配列変数名}……〕

**文例**　ERASE X,Y$

**範囲**　{配列変数名}の付け方に基づいた配列変数でなければなりません。

**解説**　DIM文が指定した配列変数を削除します。この結果フリーメモリーが増えます。ERASE文で削除した配列変数名はDIM文で再定義することができます。{配列変数名}が正しい書式になっていなかったり、DIM文で定義されていないときは、"Illegal function call"エラーになります。

**参照**　DIM

**サンプルプログラム：**

```
10 PRINT "ﾃｲｷﾞﾏｴ     ";FRE(0)
20 DIM A$(20)
30 PRINT "ﾃｲｷﾞｺﾞ     ";FRE(0)
40 FOR J=1 TO 20
50 A$(J)="ABC"
60 NEXT J
70 PRINT "ﾃﾞｰﾀ ｾｯﾄｺﾞ";FRE(0)
80 ERASE A$
90 PRINT "ERASE ｺﾞ   ";FRE(0)
100 END
```

空き領域を配列変数の定義前、定義後、データセット、イレーズ実行後の変化をみています。

## LET　〈ステートメント〉
レット

**機能**　変数に値を代入します。

**書式**　〔LET〕{変数名} ＝ {式}

**文例**　LET X = 0

**解説**　変数に値を代入します。LETは省略しても同じ意味になります。
{式}の内容は、数値、文字列のどんな型でもかまいません。ただし、文字式の値を数値変数に、または数値式の値を文字変数に代入することはできません。

**サンプルプログラム：**

```
10 LET A$="IEJHF"
20 LET B$="OJIFJ"
30 LET C$=A$+B$
40 LET D=LEN(C$)
50 PRINT C$,D
60 END
```

文字変数のデータセット、たし算をLET文で行なっています。
40行では文字列の長さをDという変数にセットします。

定義宣言 代入

# SWAP　〈ステートメント〉

スワップ

**機能**　2つの変数の値を交換します。

**書式**　SWAP{変数}, {変数}

**文例**　SWAP P$, Q$

**範囲**　{変数}は、どんな型(整数型、単精度型、倍精度型)でもかまいません。しかし、2つの型が一致していなければなりません。

**解説**　2つの変数の値を交換します。

2つの変数の型が異なると"Type mismatch"エラーになります。

**サンプルプログラム：**

```
10 INPUT "A,B=";A,B
20 IF A<B THEN SWAP A,B
30 PRINT A;">=";B
40 END
```

入力した2つの値を比較し、大きい値をAに入れ、小さい値をBに入れます。



# SGN　〈関数〉

サイン

**機能**　正負符号を調べ、正は1、0は0、負は−1を与えます。

**書式**　SGN({数式})

**文例**　A=SGN(0.5) その結果 変数Aに1の値が入る。

**解説**　{数式}の範囲が正のときは1、0のときは0、負のときは−1を与えます。

**サンプルプログラム：**

```
10 INPUT "NUMBER=";A
20 ON SGN(A)+2 GOSUB 40,50,60
30 GOTO 10
40 PRINT "<0":RETURN
50 PRINT "=0":RETURN
60 PRINT ">0":RETURN
```

入力した値が正か負か0かをSGN関数で調べています。

# SQR 〈関数〉

## スクウエア ルート

**機能** 平方根（ルート）を与えます。

**書式** SQR(｛数式｝)

**文例** X＝SQR(5) その結果 Xは2.2360679774998（$\sqrt{5}$）となる。

**範囲** ｛数式｝の値は0以上

**解説** ｛数式｝の値の平方根を求めます。
｛数式｝はどの数値でもかまいませんが、結果はすべて倍精度になります。
｛数式｝の値が負のときは“Illegal function call” エラーになります。

**サンプルプログラム：**

```
10 INPUT J:IF J<=0 THEN 10
20 S1=SQR(J)
30 S2=J^0.5
40 S3=EXP(LOG(J)/2)
50 PRINT S1;S2;S3
60 END
```

入力したデータをもとに平方根を求めています。SQR関数を使わない平方根の値も並べて表示します。

# ABS 〈関数〉

## アブソリュート

**機能** 絶対値を与えます。

**書式** ABS(｛数式｝)

**文例** P＝ABS(−2.7) その結果 P＝2.7となる。

**範囲** 倍精度

**解説** ｛数式｝の絶対値を与えます。この｛数式｝はどのような型の数値でもよく、このとき得られる値は倍精度です。

**サンプルプログラム：**

```
10 INPUT "ｶｽﾞｦ ｲﾚﾃｸﾀﾞｻｲ";A
20 PRINT " ｿﾉﾏﾏ    ";A
30 PRINT " ｾﾞｯﾀｲﾁ ";ABS(A)
40 PRINT
50 END
```

プログラムを実行すると、数字の入力を促してきますので、適当な値を入力すると、その絶対値を表示します。

# FIX

フィックス

〈関数〉

**機能**　整数部（小数点以下を取り去った値）を与えます。

**書式**　FIX（{数式}）

**文例**　S＝FIX（−3.2）〔その結果〕S＝−3となる。

**解説**　{数式}の値の小数点以下を取り去った値を求めます。FIX（X）は小数点以下を取り去った値で、INT（X）は数式の値を越えない最大の整数値になります。

**参照**　INT

**サンプルプログラム：**

```
10 PRINT "  X  INT FIX"
20 X=1.2
30 PRINT USING"+#.# ### ###";X;INT(X);FIX(X)
40 X=-1.2
50 PRINT USING"+#.# ### ###";X;INT(X);FIX(X)
60 END
```

INT関数、FIX関数のちがいを表示しています。

# INT

インティジャー

〈関数〉

**機能**　数式の値を越えない最大の整数値を与えます。

**書式**　INT（{数式}）

**文例**　A＝INT（−3.21）〔その結果〕変数Aの値が−4となる。

**範囲**　整数型、単精度実数、倍精度実数

**解説**　{数式}の値を越えない最大の整数を与えます。

**参照**　FIX，CINT

**サンプルプログラム：**

```
10 PRINT INT(3.465),INT(2.987),INT(-1.123)
20 END
```

INT関数の結果を比べるプログラムです。

# SIN 〈関数〉

サイン

**機能** 正弦(サイン)を与えます。

**書式** SIN ({数式})

**文例** A = SIN (3.14159/180＊30) その結果 Aは0.49999961698727となる。

**解説** {数式}の値の正弦(サイン)を与えます。単位はラジアン。

{数式}は、整数値、単精度、倍精度のいずれでもかまいませんが、結果はすべて倍精度になります。

**参照** COS, TAN

**サンプルプログラム**:

```
10 PAI=3.1415926536#
20 INPUT N:IF N<1 THEN 20
30 SCREEN 2:LINE(0,92)-(255,92),7:LINE(50,0)-(50,191),7:PSET(50,92)
40 S=PAI/N:M=0
50 FOR J=S TO PAI STEP S
60  X=J*200/PAI+50:Y=92-SIN(J)*90
70  LINE-(X,Y),7:M=M+SIN(J)*S
80 NEXT J
90 PAINT(100,80),7,7
100 OPEN"GRP:" FOR OUTPUT AS #1
110 PRESET(60,110)
120 PRINT #1,"ﾒﾝｾｷ";M
130 GOTO 130
```

入力された数値のステップでSIN関数により求められた数値を画面上にプロットしてその中の面積計算をするプログラムです。

# COS 〈関数〉

コサイン

**機能** 余弦 (コサイン) を与えます。

**書式** COS ({数式})

**文例** P = COS (3.14159/180＊30) その結果 P = 0.86602562491683(COS30°) となる。

**解説** {数式}の値の余弦(コサイン)を与えます。単位はラジアン。

{数式}は、整数値、単精度、倍精度のいずれでもかまいませんが、結果はすべて倍精度になります。

**参照** SIN, TAN

**サンプルプログラム**:

```
10 SCREEN 2
20 PI#=3.1415926536#
30 LINE(0,96)-(255,96)
40 LINE(0,0)-(0,191)
50 FOR J#=0 TO 2*PI# STEP PI#/100
60 X=J#/PI#*120
70 Y=96-63*COS(J#)
80 PSET(X,Y)
90 NEXT J#
100 IF INKEY$<>" "THEN 100
110 END
```

コサインのグラフを描きます。

# TAN 〈関数〉

## タンジェント

**機能**　正接（タンジェント）を与えます。

**書式**　TAN（{数式}）

**文例**　Y＝TAN（3.14159/180＊60）［その結果］Yは1.7320472694549となる。

**解説**　{数式}の値の正接(タンジェント)を与えます。{数式}の単位はラジアン。
{数式}は整数、単精度、倍精度のいずれでもかまいませんが、結果はすべて倍精度になります。

**参照**　SIN，COS

**サンプルプログラム：**

```
10 SCREEN 0:WIDTH 40
20 PAI=3.1415926536#
30 FOR J=-PAI/4+0.5 TO PAI/4-0.5 STEP 0.5
40  PRINT TAN(J);SIN(J)/COS(J)
50 NEXT J
60 END
```

正接の値を表示するプログラムです。

# ATN 〈関数〉

## アークタンジェント

**機能**　逆正接（アークタンジェント）を与えます。

**書式**　ATN（{数式}）

**文例**　P＝ATN（0）［その結果］P＝0となる。

**解説**　{数式}の値の逆正接(アークタンジェント)を与えます。この結果の値の単位はラジアンです。与えられる範囲は $-\pi/2$ から $\pi/2$ までになります。結果は倍精度です。

**サンプルプログラム：**

```
10 FOR X=1 TO 20
20   PRINT USING"X=## ";X;
30   PRINT USING"ATN(X)=#.#### ";ATN(X)
40 NEXT X
50 END
```

アークタンジェントの値を1から20まで表示します。

# EXP
〈関数〉

イクスポーネンシャル

**機能**　eに対する指数関数の値を与えます。

**書式**　EXP(｛数式｝)

**文例**　S＝EXP(3) その結果 S＝20.085536923184($e^3$)となる。

**範囲**　｛数式｝の値は145.06286085862までです。

**解説**　自然対数の底eの｛数式｝乗を値とします。

｛数式｝の値はどのような型でもかまいませんが，与えられる値は倍精度です。

**サンプルプログラム：**

```
10 SCREEN 2
20 LINE(0,180)-(255,180)
30 LINE(127,0)-(127,191)
40 FOR X=-1 TO 1 STEP 0.05
50  X1=(X+1)*127
60  Y1=180-EXP(X)*60
70  PSET(X1,Y1)
80 NEXT X
90 GOTO 90
```

指数関数のグラフを表示しています。

# LOG
〈関数〉

ログ

**機能**　自然対数を与えます。

**書式**　LOG (｛数式｝)

**文例**　A = LOG (100) その結果 変数Aに4.605170185988($\log_e 100$)が入る。

**範囲**　｛数式｝は0より大きくなければなりません。

**解説**　｛数式｝で指定した値の自然対数(eを底とした対数)を与えます。

｛数式｝型は範囲内のどのような型でも、結果は倍精度になります。

｛数式｝の値が0以下のときは“Illegal function call”エラーになります。

**サンプルプログラム：**

```
10 SCREEN 2
20 LINE(0,92)-(255,92)
30 LINE(10,0)-(10,191)
40 FOR X=0.1 TO 2.1 STEP 0.05
50  X1=X*120
60  Y1=92-LOG(X)*60
70  PSET(X1,Y1)
80 NEXT X
90 GOTO 90
```

LOG関数のグラフを表示しています。

# RND　〈関数〉

## ランダム

**機能**　0以上1未満の乱数を与えます。

**書式**　RND（{数式}）

**文例**　A = RND(1) [その結果] 変数Aに乱数の値が入る。

**範囲**　{数式}の値は、整数、単精度実数、倍精度実数のいずれでもかまいません。

**解説**　0から1未満の乱数を発生します。

RUN文が実行されるごとに同系列の乱数になります。

発生される乱数は{数式}の値により変わります。

RND（負の数値）――値により異なる、乱数系列を与えます。
RND（0）――――――1つ前に発生した乱数を値とします。
RND（正の数値）――値に関係なく次の乱数を与えます。

通常は正の値で乱数を得ますが、乱数系列が同じため、プログラムを実行させるごとに同じ乱数が発生します。

実行時に毎回異なる乱数を得たいときは、X = RND（−TIME）を実行させ、乱数系列を変えてから、RND（正の値）で乱数を求めます。このときの変数Xはダミーとして使用します。

**サンプルプログラム：**

```
10 PRINT RND(1)
20 GOTO 10
```

0から1未満の乱数が表示されます。
CTRL → STOP で中断します。

# CDBL　〈関数〉

## コンバートダブル

**機能**　整数値、単精度実数値を倍精度実数値に変換した値を与えます。

**書式**　CDBL（{数式}）

**文例**　A# = CDBL（&H257F） [その結果] A# =9599となる

**範囲**　数式は単精度実数または整数

**解説**　{数式}の値を倍精度実数に変換します。ただし、型変換が行なわれるだけで、有効桁数は変化しませんので、与えられる値の精度は変換する前の型と同じになります。

CDBLの演算結果は、倍精度実数型変数へ代入した結果と同じです。

# CINT 〈関数〉

## コンバートインテジャー

**機能** 単精度実数値、倍精度実数値を整数値に変換した値を与えます。

**書式** CINT (｛数式｝)

**文例** P = CINT (−3.479) [その結果] P = − 3となる。

**範囲** −32768〜32767

**解説** ｛数式｝の値の小数点以下を切り捨てて整数とします。

CINT での変換は、整数型変数の代入結果と同じです。ただ、与えられる値が、−32768〜32767の範囲を越えると"Overflow"と表示されます。

**参照** CSNG, CDBL

**サンプルプログラム：**

```
10 A#=10/3
20 B#=2/7
30 C%=CINT(A#/B#)
40 PRINT A#,B#,C%,A#/B#
50 END
```

A＃，B＃は倍精度の数値です。その数値の割算を行ない、整数値に変換した結果と倍精度の結果を比較します。

# CSNG 〈関数〉

## コンバートシングル

**機能** 整数値、倍精度実数値を単精度実数値に変換した値を与えます。

**書式** CSNG (｛数式｝)

**文例** P = CSNG (1234567) [その結果] P＝1234570 となる。

**範囲** 数式を単精度実数型にしたとき、−9.99999E + 62〜9.99999E + 62となる値

**解説** ｛数式｝の値を単精度実数型に変換します。変換結果は、単精度実数型変数への代入結果と同じです。

範囲を越えると"Overflow"エラーになります。

**参照** CINT, CDBL

**サンプルプログラム：**

```
10 A#=10/3
20 B#=2/7
30 C!=CSNG(A#/B#)
40 PRINT A#,B#,C!,A#/B#
50 END
```

A＃,B＃は倍精度の数値です。A＃／B＃の割算の結果を、A＃,B＃の次に単精度に変換した場合と、しない場合を表示します。

# ASC 〈関数〉

## アスキー

**機能** 文字列の最初の文字のキャラクターコードを与えます。

**書式** ASC(｛文字列｝)

**文例** P=ASC("9") [その結果] P=57となる。

**範囲** キャラクターコードの0～255

**解説** ｛文字式｝の最初の文字のキャラクターコードを与えます。文字のキャラクターコードの対応についてはキャラクターコード表を参照してください。

｛文字式｝がヌルストリング(長さが0の文字列)のときは"Illegal function call"エラーになります。ただし、引用符(")をデータとして扱うことはできません。

**参考** CHR$,付録「キャラクターコード表」

**サンプルプログラム：**

```
10 FOR J=1 TO 5
20   D$=INPUT$(1)
30   PRINT ASC(D$)
40 NEXT J
50 END
```

キーボードから入力した文字に対応するアスキーコードを表示します。5回くりかえします。



# CHR$ 〈関数〉

## キャラクターダラー

**機能** 指定したキャラクターコードに対応する文字を与えます。

**書式** CHR$(｛数式｝)

**文例** A$ = CHR$(&H46) [その結果] A$に"F"が入る。

**範囲** 0～255

**解説** ｛数式｝の値のキャラクターコードを持つ文字を与えます。

｛数式｝の値が32～255の場合は、そのキャラクターコードに対応する文字を得ることができます。｛数式｝の値が0～31(コントロールコード)のときは、そのコードの持つ機能が実行され、キャラクターコードに対応する文字の表示はされません。なお、｛数式｝の値が0～255を越えると"Illegal function call"エラーとなります。

**参照** ASC、付録「キャラクターコード表」

**サンプルプログラム：**

```
10 FOR J=1 TO 31
20   PRINT CHR$(1)+CHR$(J+64);
30 NEXT J
40 FOR J=32 TO 255
50   PRINT CHR$(J);
60 NEXT J
70 END
```

10行～30行はグラフィックを表示し、40行～60行は通常の文字を表示します。

# STR$ 〈関数〉

## エスティーアールダラー

**機能** 数値を文字列に変換します。

**書式** STR$（{数式}）

**文例** A$＝STR$（１２３） その結果 A$に数式１２３を文字列␣１２３として代入する。

**範囲** {数式}は、整数、単精度、倍精度実数のいずれの数値型でも可能です。

**解説** {数式}で示される数値を10進表記の文字列に変換します。

**参照** VAL，HEX$, OCT$, BIN$

**サンプルプログラム：**

```
10 INPUT A
20 A$=STR$(A)
30 A$=RIGHT$("     "+A$,5)
40 PRINT A$
50 END
```

入力した数値(最大５桁)を文字に直して表示します。
30行の空白(スペース)は５つです。

# BIN$ 〈関数〉

## バイナリーダラー

**機能** 数値を２進表記の文字列に変換して、その結果を与えます。

**書式** BIN$（{数式}）

**文例** S$＝BIN$(8) その結果 S$には"1000"が与えられる。

**範囲** －32768～65535

**解説** {数式}の値を２進表記の文字列に変換します。
変換結果は文字列となり、先頭の不用な０は取り除かれます。(ゼロサプレス)
{数式}が範囲を越えると"Overflow"エラーになります。

**参照** VAL, OCT$, HEX$

**サンプルプログラム：**

```
10 FOR J=0 TO 15
20   PRINT RIGHT$("0000000000000000"+BIN$(2^J),16)
30 NEXT J
40 END
```

０から15までを２進数で表示します。

文字列操作

# OCT$

〈関数〉

オクタルダラー

**機能** 数値を8進数に変換して、その結果を文字として与えます。

**書式** OCT$ ({数式})

**文例** A$ = OCT$ (100) その結果 A$には"144"が与えられる。

**範囲** {数式}の値は−32768〜65535

**解説** {数式}の値を8進表記の文字列に変換し、その結果を与えます。
変換結果の文字列はゼロサプレス(はじめの不用な0を取り除く)されています。

**サンプルプログラム：**

```
10 FOR J=0 TO 255
20 PRINT USING"10:### 16:&& 8:& &";J;HEX$(J);OCT$(J)
30 NEXT J
40 END
```

0〜255までの数値を16進数、8進数で表示します。



# HEX$

〈関数〉

ヘキサダラー

**機能** 10進数を16進数の文字列に変換して、その結果を与えます。

**書式** HEX$ ({数式})

**文例** A$ = HEX$ (33) その結果 A$には21が与えられる。

**範囲** −32768〜65535

**解説** {数式}の値を16進表記の文字列に変換し、その結果を与えます。変換結果の文字列はゼロサプレス(先頭の不用な0を取り除く)されます。

**参照** VAL, BIN$, OCT$

**サンプルプログラム：**

```
10 FOR J%=-32768! TO 32767
20   PRINT J%,HEX$(J%)
30 NEXT J%
40 END
```

10進数とそれに対応する16進数を表示します。

# VAL 〈関数〉

バリュー

**機能** 文字列の表わす内容を数値に変換します。

**書式** VAL ({文字列})

**文例** B = VAL ("&H" + "1F") [その結果] Bには31(&H1F)の数値が入る。

**解説** {文字列}で表わされる内容を数値に変換します。
{文字列}の最初の文字が+、−、&または数字以外のときは、VALの値は0になります。

**サンプルプログラム：**

```
10 INPUT A$
20 A=VAL(A$)
30 PRINT A
40 END
```

文字列で入力したデータを数値に変換し、表示します。

文字列操作

# STRING$ 〈関数〉

ストリングダラー

**機能** 指定された文字を指定された数だけ与えます。

**書式** STRING$ ({式}, | {文字式} / {数式} |)

**文例** A$ = STRING$ (3,76) [その結果] A$にキャラクターコード76番の文字3個を代入する。

**範囲** {式}の値、{数式}の値、ともに0～255までです。

**解説** 指定する文字列を{式}の値の数だけ与えます。
{文字式}の場合は、最初の1文字だけが有効で、その文字を{式}の数だけ与えます。
{数式}の場合は、その値をキャラクターコードとして、そのコードに対応する文字を与えます。（付録キャラクターコード参照）

**サンプルプログラム：**

```
10 PRINT STRING$( 15,"+" )
20 PRINT STRING$( 5,"ABC")
30 END
```

STRING$関数を使って文字列を表示します。

# INSTR 〈関数〉

インストリング

**機能** 文字列の中から指定された文字列をさがして、その文字の位置を与えます。

**書式** INSTR (〔{数式},〕{文字列1}, {文字列2})

**文例** A＝INSTR ("１２３ａｂｃ", "ａ") [その結果] "１２３ａｂｃ"に含まれる"ａ"の左からの位置「４」が変数Aに与えられる。

**範囲** {数式} はさがしはじめる位置を0～255の範囲で指定します。

**解説** {文字列1} の中から {文字列2} があるかどうかをチェックし、あれば発見した位置を、なければ0を値として与えます。

もし {数式} を省略した場合は {文字列1} の左端からチェックします。

{文字列1}、{文字列2} は文字式で指定します。

{文字列2} の長さが {文字列1} より大きかったり、{文字列1} がヌルストリングであれば0を値として与えます。

{数式}、{文字列1}、{文字列2} の値が適切でなければ "Illegal function call" エラーとなります。

**参照** LEN

**サンプルプログラム：**

```
10 INPUT A$
20 A=INSTR$(A$," " )
30 IF A>0 THEN PRINT MID$(A$,A)
40 END
```

文字列を入力して、その文字列の中に空白(スペース)があれば、その文字以降を表示します。

# LEN 〈関数〉

レングス

**機能** 文字列の総文字数を与えます。

**書式** LEN ({文字列})

**文例** A = LEN ("ＡＢ") [その結果] 変数Aには2が入る。

**範囲** {文字列} の長さは0～255文字

**解説** {文字列} の長さを与えます。

グラフィック記号のグラフィックキャラクターヘッダ (&H01) も1文字と数えるので、グラフィック記号1文字の文字数は2となります。

**サンプルプログラム：**

```
10 INPUT A$
20 PRINT LEN(A$)
30 END
```

キーボードから入力した文字列の長さを表示します。

# LEFT$ 〈関数〉

レフトダラー

**機能**　文字列の左側から指定された長さの文字列を与えます。

**書式**　LEFT$({文字列},{数式})

**文例**　A$ = LEFT$ ("CASIO", 2) その結果 変数A$には "CA" が入る。

**範囲**　{文字列}は255文字まで{数式}は0～255

**解説**　{文字列}の左端から数式分だけを取り出します。
{数式}が0のときはヌルストリングになります。
{数式}が文字列の長さ以上のときの結果は、{文字列}のすべてです。
グラフィック記号のグラフィックキャラクターヘッダ（&H01）も1文字として数えられるので、グラフィック記号の文字数は、合計2文字となります。
A$ = LEFT$ ("時分秒", 2) その結果変数A$には"時"が入ります。

**サンプルプログラム：**

```
10 INPUT A$
20 FOR J=1 TO LEN(A$)
30  PRINT LEFT$(A$,J)
40 NEXT J
50 END
```

入力した文字を左から1文字、2文字……と表示します。

# RIGHT$ 〈関数〉

ライトダラー

**機能**　文字列の右側から指定された長さの文字列を与えます。

**書式**　RIGHT$ ({文字列},{数式})

**文例**　A$ = RIGHT$ ("CASIO", 3) その結果 A$に "SIO" が入る。

**範囲**　{文字列}は255字まで。{数式}は0～255

**解説**　{文字列}の右側から{数式}分だけの文字列を与えます。
{数式}が0のときは結果はヌルストリングになります。
{数式}が{文字列}の長さよりも大きいときは{文字列}のすべてが結果になります。
グラフィック記号のグラフィックキャラクターヘッダ（&H01）も1文字として数えるのでグラフィック記号1文字は、合計2文字として計算されます。
(例)　A$=RIGHT$ ("日月火", 2) その結果A$に "火" のみが入る。

**参照**　LEFT$, MID$

**サンプルプログラム：**

```
10 INPUT A$
20 FOR J=1 TO LEN(A$)
30  PRINT RIGHT$(A$,J)
40 NEXT J
50 END
```

入力した文字を右から1文字、2文字、3文字……と表示します。

# MID$ 〈ステートメント〉

ミッドダラー

**機能** 文字変数の一部をおき換えます。

**書式** MID$（{文字変数}，{式1}〔,{式2}〕）＝{文字列}

**文例** MID$（P$，2，3）＝"XYZ"

**範囲** {文字列}は文字式、{式1}、{式2}は数式

**解説** {文字変数}の{式1}番目の文字から{式2}個の文字を、{文字列}の最初から{式2}個の文字列におき換えます。

{式2}を省略した場合、または{文字列}の文字数より多く指定した場合は、{文字列}のすべての文字とおき換わります。

{式1}の値は{文字変数}の文字数より大きかったり、0以下ではいけません。

このときは"Illegal function call"エラーになります。

**参照** MID$（関数）

**サンプルプログラム：**

```
10 A$="CASIO 16K MSX"
20 PRINT A$
30 N=INSTR(A$,"16")
40 MID$(A$,N,2)="32"
50 PRINT A$
60 END
```

CASIO 16K MSXの16を32に変更するプログラムです。

# MID$ 〈関数〉

ミッドダラー

**機能** 文字列のなかから指定された長さの文字列を与えます。

**書式** MID$（{文字列}，{式1}〔，{式2}〕）

**文例** A$ = MID$（"ABCDEF"，4，2）［その結果］A$には文字列の左から4番目の2文字DEが入る。

**範囲** {式1}は1～255、{式2}は0～255

**解説** {文字列}のなかから必要な文字列を取り出します。{式1}番目の文字から{式2}文字分の文字列を取り出します。

{式2}を省略した場合や、{文字列}の{式1}番目の文字から右の文字数が{式2}より小さい場合は、{文字列}の{式1}番目より右すべての文字列が取り出されます。

{文字列}の文字数が{式1}より小さいときは、取り出す文字列はヌルストリングとなります。

値が適切でないときは"Illegal function call"エラーとなります。

**参照** RIGHT$，LEFT$，MID$（ステートメント）

**サンプルプログラム：**

```
10 INPUT A$
20 FOR J=1 TO LEN(A$)
30 PRINT MID$(A$,J,1);" ";
40 NEXT J
50 END
```

入力した文字列をひと文字ずつ表示するプログラムです。

# SPACE$ 〈関数〉

スペースダラー

**機能** 指定された長さの空白文字列を与えます。

**書式** SPACE$（{数式}）

**文例** A$ =SPACE$（5）［その結果］"␣␣␣␣␣" がA$に入る。

**範囲** {数式}は、0～255

**解説** {数式}の値だけ空白の文字列を与えます。

**参照** SPC

**サンプルプログラム：**

```
10 FOR J=1 TO 10
20  A$="A"+SPACE$(J)+"A"
30  PRINT A$
40 NEXT J
50 END
```

"A"と"A"の間に空白を入れて表示します。

文字列操作

## SPC 〈関数〉
スペース

**機 能** 画面やプリンタに指定された数だけ空白を出力します。

**書 式** SPC({数式})

**文 例** PRINT "AB"; SPC(3); "C" [その結果] AB␣␣␣C と画面に表示される。

**範 囲** {数式}の値は0～255

**解 説** PRINTおよび LPRINT 文で{数式}の値だけ空白を出力します。
PRINTおよび LPRINT 文など出力に関係するステートメントの中だけで使用できる関数で、LET文などの代入文では使用できません。

**参 照** SPACE$, TAB

**サンプルプログラム：**

```
10 FOR J=1 TO 10
20 PRINT"ABCD";SPC(J);"EFG"
30 NEXT J
40 END
```

"ABCD"と"EFG"の間に空白を入れて表示します。



## GOTO 〈ステートメント〉
ゴートゥー

**機能** 指定された行へジャンプします。

**書式** (ア) GOTO {行番号}
(イ) GO TO {行番号}

**文例** GOTO 100

**解説** プログラムの流れを無条件に指定した{行番号}にジャンプする命令です。書式の(ア)、(イ)は全く同じ働きをしますが、"GO"と"TO"の間のスペースは1個のみです。2個以上のときは、"Syntax error"になります。指定した{行番号}がないときは"Undefind line number"エラーになります。
GOTO文をダイレクトモードで実行すると指定した行からプログラムを実行します。ただし、このときはRUN命令と異なり、変数の初期化やオープンしているファイルのクローズなどは行ないません。

**サンプルプログラム：**

```
10 PRINT "ABCDE",
20 GOTO 10
```

プログラムを実行するとGOTO文によりいつまでも"ABCDE"を表示します。
CTRL → STOP でプログラムは停止します。

# GOSUB　〈ステートメント〉

ゴーサブ

**機能**　サブルーチンを呼び出します。

**書式**　GOSUB｛行番号｝

　　　　　∫

　　　　RETURN〔｛行番号｝〕

**文例**　GOSUB 200

**解説**　｛行番号｝からはじまるサブルーチンを GOSUB 文によって実行します。その後 RETURN 文が実行されると、GOSUB 文の次の文へ戻ります。RETURN 文に行番号を付けると、その行に戻ります。

サブルーチンとは独立したひとつのプログラムで、GOSUB で指定する｛行番号｝から、RETURN 文までの一連の命令のことをいいます。

ひとつのサブルーチンの中で GOSUB 文を使うこともできます。これはメモリーのスタック領域の容量が許すかぎり行なうことができます。もし、スタック領域がたりなくなった場合には、"Out of memory"エラーになります。

指定した｛行番号｝がない場合は、"Undefined line number"エラーになります。

サブルーチンの中で、CLEAR文や MAXFILES文を実行すると、"RETURN without GOSUB"エラーとなり RETURN文で戻ることができなくなります。

**参照**　RETURN

**サンプルプログラム：**

```
10 GOSUB 60
20 D=C
30 GOSUB 60
40 PRINT C*D
50 END
60 INPUT A,B
70 C=A*B
80 RETURN
```

60行～80行がサブルーチンです。このサブルーチンはAとBの値を入力し、入力した2つの値のかけ算をした結果をCに代入します。その後メインルーチンに戻ります。

分岐

# RETURN 〈ステートメント〉

リターン

**機能** サブルーチンから復帰します。

**書式** RETURN〔{行番号}〕

**文例** RETURN 200

**範囲** {行番号}は0～65529

**解説** サブルーチン処理を終え、GOSUB 文の次の文を実行します。{行番号}を指定すると、その行番号にジャンプします。
割り込み処理ルーチン(ON STOP GOSUB, ON STRIG GOSUB…)などのときは、割り込みがかかったときに中断されたプログラムが実行されます。
サブルーチンの中に複数の RETURN 文があってもかまいません。
GOSUB 文が実行されずに RETURN 文が実行されると“RETURN without GOSUB”エラーになります。

**参照** GOSUB, ON～GOSUB



# IF～THEN～ELSE/IF～GOTO～ELSE 〈ステートメント〉

イフ ゼン エルス/イフ ゴートゥー エルス

**機能** 条件判断を行ないます。

**書式**

```
IF {論理式} | THEN | {文}     | | [ELSE | {文}     | ]
            |      | {行番号} | |       | {行番号} |
            | GOTO   {行番号} |
```

**文例** IF X=0 THEN 100 ELSE 200

**解説** 条件判断を行なうもので、条件が成立するならばTHEN 文または GOTO 文を実行し、条件が成立しないときはELSE 文を実行します。
条件が成立しない時で ELSE 文がない場合は、IF 文の次の行を実行します。
{条件}は式で表します。

●数値や文字列の大小比較

数式や文字で表した2つの数値や文字列を、比較演算子(=、<、>など)を使って比較します。

(例)

```
IF A<100 THEN 処理1
```

(変数Aの内容が100より小さければ(真)、処理1を行ないます。)

```
IF A$="TEST" THEN 100
```

(変数 A$ の内容が"TEST"なら(真)、行番号100へジャンプします。)

比較した結果、条件が成立しない状態を“偽”といい、{条件}が持つ値は0になります。このときELSE文があればELSE文を実行し、ELSE文がなければ、次の行を実行します。
比較演算子には次のような種類があります。

| 比較演算子 | 意　　味 | 例 |
|---|---|---|
| {式1} = {式2} | {式1}と{式2}が等しい。 | A$=B$ |
| {式1}<>{式2} | {式1}と{式2}が等しくない。 | A+B<>C |
| {式1} < {式2} | {式1}が{式2}より小さい。 | A<100 |
| {式1} > {式2} | {式1}が{式2}より大きい。 | A>B+C |
| {式1}<={式2}<br>{式1}=<{式2} | {式1}が{式2}と等しいか、{式2}より小さい。 | A$<=B$ |
| {式1}>={式2}<br>{式1}=>{式2} | {式1}が{式2}と等しいか、{式2}より大きい。 | A$=>“A” |

数値の大小の比較は代表的に行なわれます。また文字列の大小比較は、それぞれの文字列の左端から右端にかけて順番に比較してゆき、全部が同じなら2つの文字列は等しく、途中で等しくない文字があれば、その文字のキャラクターコードの大きい方の文字列が大きいということになります。(キャラクターコード表参照)
文字列の片方が短かくて比較が途中で終る場合は、短かい文字列の方が小さいということになります。なお、文字列中の空白にも意味があります。

●数値が0であるか否かの判断

数式の値が0であるか否かを判断します。

(例)

```
IF A=0 THEN 250 ELSE 100※
```

(変数Aの値が0ならば行番号250へ、それ以外は100へジャンプします。)

```
IF A+B<>0 THEN 300
```

(変数A、Bの和が0以外ならば行番号300へジャンプします。)

●複数条件の判断

複数個の条件を論理的に演算子(NOT, AND, OR)を使って判断します。

(例)

```
IF X<0 OR X>99 THEN 500
```

(Xが負、または99より大きければ行番号500へジャンプします。)

```
IF X>0 AND X<100 THEN 処理2
```

(Xが正で、なおかつ100より小さければ処理2を行ないます。)

```
IF NOT(A=0) THEN 50
```

(Aが0でなければ、行番号50へジャンプします。)

※IF A THEN 100 ELSE 250としても同じ条件判断を行ないます。

| | 論理演算子 | 意味 | 例 |
|---|---|---|---|
| 高<br>↑<br>優先順位<br>↓<br>低 | NOT {条件}<br>{条件1} AND {条件2}<br><br>{条件1} OR {条件2} | {条件}が偽である<br>{条件1}と{条件2}が共に真である。<br>{条件1}{条件2}のうち少なくとも、どちらかひとつが真である。 | NOT (A = 0)<br>A = 0 AND B = 0<br><br>A = 0 OR B = 0 |

比較の優先順位を変えたいときはカッコ( )を使います。

(例)

IF A=0 AND(B=0 OR C=0) THEN 300

(A = 0でB = 0、またはA = 0でC = 0のとき行番号300へジャンプします。)

IF ~ THEN ~ ELSE文では、THEN文やELSE文の中へ、さらに別のIF文を置いて入れ子（ネスティング）にすることができます。IF文は何重にも入れ子にすることができますが、1行（255文字）に書ける範囲に限られます。

**サンプルプログラム：**

```
10 INPUT A
20 IF INT(A/2)*2=A THEN PRINT "ｸﾞｳｽｳ" ELSE PRINT "ｷｽｳ"
30 END
```

入力した数字が偶数か奇数かを判断しています。

# FOR〜NEXT　〈ステートメント〉

フォーネクスト

**機能**　FOR文からNEXT文までを条件にしたがって繰り返し実行します。

**書式**　FOR〔{変数名}＝{初期値}　TO{終値}〔STEP{増分}〕

NEXT〔〔{変数名}〕〔,{変数名}〕〕

**文例**　FOR　J＝1　TO　100 STEP　10

〜

NEXT J

**範囲**　{変数名}は、整数、単精度、倍精度型のいずれかを指定します。

**解説**　{変数名}は繰り返しの制御変数です。

{初期値}は制御変数の初期値になります。終値は制御変数の最終値になります。制御変数は、1回実行するたびにSTEPで指定された増分だけ増えます。(増分が負のときは引かれていきます。)このように加えたり引いたりした結果、制御変数が終値を越えたかどうかチェックし、越えていなければ、FOR〜NEXTの間にはさまれた命令を実行します。越えていれば、NEXT文の次の文が実行されます。これをFORループと呼びます。{増分}を省略すると制御変数は1ずつ増えていきます。{増分}が負の場合は終値が初期値以下でなければなりません。ループの本体は初期値が終値を越えている場合でも一回は実行します。

FOR〜NEXTの間に、FOR〜NEXT(制御変数は異なる)を入れることができます。これを、入れ子(ネスティング)と呼び、内側のループのNEXT文は外側のNEXT文の前になければいけません。

```
┌FOR J=1 TO 10
│┌FOR K=1 TO 10
││    〜
│└NEXT K
└NEXT J
```

ネスティングの数はメモリーの空き領域の大きさに制限を受けます。

NEXT文で変数をつなぐことができますが、省略したときは、最も近くにあるFOR文に対応します。

```
FOR J=1 TO 10
    FOR K=1 TO 10
          〜
    NEXT K ┐
           ├ ⇒NEXT K, J
    NEXT J ┘
```

FOR文がなくてNEXT文があったときは“NEXT without FOR”エラーになります。

**サンプルプログラム：**

```
10 FOR J=1 TO 10
20   FOR K=1 TO 10
30     PRINT J,K
40   NEXT K
50 NEXT J
60 END
```

2重ループのFOR〜NEXT文です。J,Kの値に注意してください。

分岐

# ON GOTO/ON GOSUB 〈ステートメント〉

オン ゴートゥー／オン ゴーサブ

**機能** 指定されたいずれかの行へジャンプします。

**書式** ON {式} GOTO {行番号} 〔, {行番号} ……〕
ON {式} GOSUB {行番号} 〔, {行番号} ……〕

**文例** ON X GOTO 100, 200, 300
ON Y GOSUB 100, 200, 300

**解説** {式} の値に対応して、GOTO 文または GOSUB 文を実行します。たとえば、{式} の値が 2 のときは 2 番目の行番号に対応します。
{式} の値が負のときは"Illegal function call"エラーが起こりますが、値が 0 または {行番号} の個数より大きくなった場合には、ON GOSUB/ON GOTO 文の次の行を実行します。
行番号がないときは"Undefined line number"エラーになります。

**参照** GOSUB, GOTO

**サンプルプログラム**：

```
10 A=INT(RND(1)*3)+1
20 ON A GOTO 40,50,60
30 GOTO 10
40 PRINT " ABC";:GOTO 30
50 PRINT " DEF";:GOTO 30
60 PRINT " GHI";:GOTO 30
```

10行で A の値を 1 から 3 まで変えています。その値により、表示する文字を変更します。
CTRL ＋ STOP で実行を中断します。



# STOP 〈ステートメント〉

ストップ

**機能** プログラムの実行を停止して、コマンド待ちになります。

**書式** STOP

**文例** STOP

**解説** プログラムを中断し、コマンド状態に戻ります。このとき画面はテキストモードになってしまうため、グラフィックモード中にSTOP文を使いCONTする場合は注意してください。
この命令でプログラムが中断すると "Break in yyyyy" (yyyyy は中断した行番号) が表示されます。
なお、停止したプログラムはCONT コマンドで再開することができます。
END文と異なり、ファイルはクローズされません。

**参照** CONT

**サンプルプログラム**：

```
10 PRINT "STOP!!!"
20 STOP
30 PRINT "CONT!!!"
40 END
```

"STOP!!!" を表示した後、プログラムは中断しますので、CONT ↵ と操作します。その結果CONT !!! を表示しプログラムが終了します。

# END　<ステートメント>

エンド

**機能**　プログラムの終了を宣言します。

**書式**　END

**文例**　END

**解説**　プログラムの実行を終了し、すべてのファイルは閉じられて、コマンドレベルに戻ります。END文はプログラムの実行を終了させたい所に何カ所でも置くことができます。
なお、プログラムの最後のEND文は省略することができますが、このときファイルはクローズされません。

**サンプルプログラム：**

```
10 INPUT "A=";A
20 INPUT "B=";B
30 IF A>B THEN 70
40 IF A<B THEN 90
50 PRINT "A=B"
60 END
70 PRINT "A>B"
80 END
90 PRINT "A<B"
100 END
```

AとBの値を入力し、比較結果を表示したあとに、END文を実行します。

分岐

# CALL　<ステートメント>

コール

**機能**　拡張ステートメントを呼び出します。

**書式**　CALL {拡張ステートメント名}〔({引数}〔,{引数}……〕)〕

**文例**　CALL TALK ("A")

**解説**　拡張ROMカートリッジでサポートされる拡張ステートメントを呼び出します。
"CALL"の代りに"_"（アンダーバー：キャラクターコード&H5F）も使用できます。
拡張ステートメントが定義されていないときは"Syntax error"が表示されます。拡張ステートメントに関しては、各ROMカートリッジの説明書や、MSXのソフトウェア仕様に関する書籍をご覧ください。
アンダーバーは［かな］キーがロックされていないとき［SHIFT］→［_］と操作します。

**参照**　付録「キャラクターコード表」

# ON ERROR GOTO 〈ステートメント〉

オン エラー ゴートゥー

**機能** エラー割り込み機能を可能にし、エラー処理ルーチンの開始行を定義します。

**書式** ON ERROR GOTO {行番号}

**文例** ON ERROR GOTO 100

**解説** この命令を実行しておくと、エラーが発生したときに割り込みがかかり、指定したエラー処理ルーチンへジャンプします。

通常、エラーが発生すると、BASICで用意しているエラーメッセージが表示され、プログラムの実行を終了し、コマンドレベルに戻りますが、この命令を実行してあると、エラーが発生したときにBASICで用意しているエラー処理は行なわれず、{行番号}で指定したエラー処理ルーチンへジャンプします。

ON ERROR GOTO 0を実行すると、ON ERROR GOTO {行番号}での定義を無効とし、以降、エラーが発生すれば、BASICで用意しているエラー処理が行なわれます。エラー処理ルーチンの実行中に発生したエラーには効力がありません。

この文実行後はコマンドレベルに戻ってから発生したエラーに対しても、割り込みがかかり、指定した行のエラー処理ルーチンからプログラムが実行します。ON ERROR GOTO文による割り込みは、別のON ERROR GOTO文か、RUN命令、CLEAR文を実行するまで禁止されません。

したがってプログラムの終了前に、ON ERROR GOTO 0を実行しておくことをおすすめします。

もし{行番号}の行がプログラム上に存在しなければ“Undefined line number”エラーになります。

# ERROR 〈ステートメント〉

エラー

**機能** プログラムをエラー状態にする。

**書式** ERROR {エラーコード}

**文例** ERROR 100

**範囲** {エラーコード}は0〜255

**解説** 付録のエラーメッセージ表に示されたエラーコードを指定すると、そのエラーメッセージを表示して、コマンドレベルに戻ります。

ON ERROR GOTO文といっしょに使うとこの文を実行後、指定したエラー処理ルーチンへジャンプし、ERRには指定したエラーコードがセットされます。

**参照** ON ERROR GOTO〜, ERR, ERL

**サンプルプログラム：**

```
10 ON ERROR GOTO 40
20 ERROR 5
30 PRINT"ABC":END
40 RESUME NEXT
```

20行でエラーを発生させていますが、RESUME NEXT文でエラーの行の次からプログラムが実行されます。

# ERL/ERR 〈システム変数〉

## エラーライン/エラーナンバー

**機能** 発生したエラーのエラーコードおよびエラーの発生した行番号を与えます。

**書式** ERL
ERR

**文例** X＝ERL
Y＝ERR

**解説** プログラム実行中にエラーがおきたときの行番号(ERL)、エラーコード番号(ERR)を値にします。ダイレクトモードのときの行番号は65535になります。ON ERROR GOTO文によって指定したエラー処理ルーチンで使うことが一般的です。
ERLをIF文中で使うときには必ず左辺におきます。(IF ERL＝10～) そうすれば、RENUMコマンドを実行したときに、右辺の値が行番号とみなされ、つけなおされるためです。ERL、ERRには値を代入することができません。これをすると "Syntax error" となります。

**参照** ON ERROR GOTO～



# RESUME 〈ステートメント〉

## リジューム

**機能** エラー回復処理終了後、プログラムの実行を再開します。

**書式** RESUME |〔0〕/ NEXT / {行番号}|

**文例** RESUME 0
RESUME NEXT
RESUME 1000

**範囲** {行番号}は、0～65529

**解説** ON ERROR GOTO でのエラー処理ルーチン中で使用し、RESUME文の実行によってエラー処理ルーチンから復帰します。
RESUME または RESUME 0 はエラーの原因となった文からプログラムを実行し、RESUME NEXT は、エラーの原因となった文のすぐ次の文からプログラムを実行します。
RESUME {行番号} は、{行番号} で指定した行から再開されます。

# INTERVAL ON/OFF/STOP 〈ステートメント〉

インターバル オン/オフ/ストップ

**機能** タイマー割り込みを許可、禁止、または保留します。

**書式** (1) INTERVAL ON

(2) INTERVAL OFF

(3) INTERVAL STOP

**文例** INTERVAL ON

**解説** ON INTERVAL GOSUB 文でタイマー割り込み時間と割り込み処理ルーチンの先頭行を定義します。

INTERVAL ON 文はタイマー割り込みを許可します。この命令を実行しておくと、ON INTERVAL GOSUB 文で設定した時間ごとに割り込みがかかり、実行中のプログラムを中断してタイマー割り込み処理ルーチンが実行されます。

INTERVAL OFF 文はタイマー割り込みを禁止します。この命令を実行しておくと、ON INTERVAL GOSUB 文で設定した時間が経過しても割り込みはかかりません。

INTERVAL STOP 文はタイマー割り込みを保留します。この命令を実行しておくと、以後にかかったタイマー割り込みは、INTERVAL ON 文を実行するまで保留され、INTERVAL ON を実行すると、直ちにタイマー割り込み処理ルーチンが実行されます。この命令は INTERVAL ON 状態で実行される場合のみ有効です。

**参照** ON INTERVAL GOSUB

**サンプルプログラム**：

```
10 CLS
20 IN INTERVAL=60 GOSUB 60
30 X=60
40 INTERVAL ON
50 GOTO 50
60 INTERVAL OFF:X=X-1:IF X=0 THEN PLAY"O4CEG05C":END
70 LOCATE 10,10:PRINT USING"##";X
80 INTERVAL ON:RETURN
```

一定時間ごとに割り込みが発生するプログラムです。

割込み

# ON INTERVAL GOSUB 〈ステートメント〉

オン インターバル ゴーサブ

**機能**　タイマー割り込み条件を設定し、割り込みルーチンの開始行を定義します。

**書式**　ON INTERVAL = {時間} GOSUB {行番号}

**文例**　ON INTERVAL = 60　GOSUB　200

**範囲**　{時間}は1/60秒単位で、1～65535までの数値、{行番号}は割り込み処理ルーチンの開始行番号で0～65529の整数です。

**解説**　この命令は、割り込みがかかる時間間隔の設定をし、その間隔ごとに割り込みがかかり、割り込み処理ルーチンが実行されます。(INTERVAL ON のとき)

割り込み処理ルーチンからの復帰は、RETURN 文を使います。

割り込み処理ルーチンを実行している間、自動的に INTERVAL STOP 状態になり、RETURN 文実行とともに INTERVAL ON 状態に戻ります。

(注)この命令は割り込みの定義をするだけで、実行したときに割り込みがかかるわけではありません。

**参照**　INTERVAL ON/OFF/STOP

**サンプルプログラム：**

```
10 J=0
20 ON INTERVAL=1 GOSUB 50
30 INTERVAL ON
40 GOTO 40
50 J=J+1
60 PRINT "TIME=";J
70 RETURN
```

時間を表示するプログラムです。
時間は1/60秒単位で増加します。

割込み

# KEY(n)ON/OFF/STOP 〈ステートメント〉

**キー オン/オフ/ストップ**

**機能** ファンクションキーによる割り込みを許可、禁止、または保留します。

**書式** (1) KEY〔({キー番号})〕ON

(2) KEY〔({キー番号})〕OFF

(3) KEY〔({キー番号})〕STOP

**文例** KEY(2) ON

**範囲** {キー番号}は1～10

**解説** ファンクションキーによる割り込みを制御します。

KEY(n) ONは、ファンクションキーn(nはキー番号)による割り込みを許可します。この文を実行したあとにファンクションキーを押すと、ON KEY GOSUB文で定義した割り込み処理ルーチンを実行します。

KEY(n) OFFは割り込みを禁止します。

KEY(n) STOPは割り込みを保留にします。この命令を実行するとファンクションキーの割り込みは実行されませんが、そのあとKEY(n) ON命令が実行されると、割り込み処理ルーチンを実行します。ただし KEY(n) STOPは KEY(n) ONの実行を保留する命令であるため、 KEY(n) ON状態でなければ保留されずに無効となります。

**参照** ON KEY GOSUB

# ON KEY GOSUB 〈ステートメント〉

**オン キー ゴーサブ**

**機能** ファンクションキーによる割り込み処理ルーチンの開始行を定義します。

**書式** ON KEY GOSUB {行番号}〔,{行番号}……〕

**文例** ON KEY GOSUB 100, 200, 300

**解説** ファンクションキーが押されたときにジャンプする処理ルーチンの開始行を定義します。ファンクションキーが押されて割り込みがかかったときに、どの割り込み処理ルーチンにジャンプするかを定義します。以後、KEY(n) ON状態にしておけば、ファンクションキーが押されるたびに割り込み処理ルーチンが実行されます。

割り込み処理ルーチンからの復帰はRETURN文を使います。

(注)この命令は割り込みの定義をするだけで、実行したときに割り込みがかかるわけではありません。

**参照** KEY(n)ON/OFF/STOP

# ON SPRITE GOSUB 〈ステートメント〉

オン スプライト ゴーサブ

**機能** スプライトが重なった場合、ジャンプする割り込み処理ルーチンの開始行を定義します。

**書式** ON SPRITE GOSUB {行番号}

**文例** ON SPRITE GOSUB 700

**解説** SPRITE ON 状態で、PUT SPRITE で画面に描いたスプライトが、他のスプライトと重なったときに割り込みがかかり、{行番号}から始まる割り込み処理ルーチンが実行されます。

割り込み処理ルーチンからの復帰は、RETURN 文を使います。

割り込み処理ルーチンを実行している間、自動的に SPRITE STOP 状態になり、RETURN 文実行とともに SPRITE ON 状態になります。

**参照** SPRITE ON/OFF/STOP, PUT SPRITE

**サンプルプログラム**：

```
10 RESTORE
20 SCREEN 2,1
30 ON SPRITE GOSUB 180
40 SPRITE ON
50 GOSUB 150 : SPRITE$(0)=A$
60 GOSUB 150 : SPRITE$(1)=A$
70 DEF FNR(X)=INT(RND(1)*X)
80 X1=FNR(240):X2=FNR(240)
90 Y1=FNR(180):Y2=FNR(180)
100 PUT SPRITE 0,(X1,Y1)
110 PUT SPRITE 1,(X2,Y2)
120 GOTO 80
130 DATA 18,18,18,FF,FF,18,18,18
140 DATA 18,24,42,FF,FF,42,24,18
150 A$="":FOR J=1 TO 8
160 READ B$:A$=A$+CHR$(VAL("&H"+B$))
170 NEXT J:RETURN
180 SPRITE OFF
190 PLAY "O4G05C"
200 FOR J=1 TO 3000:NEXT J
210 END
```

2つのスプライトを画面上に表示し、2つのスプライトが重なったときに音楽を鳴らして処理を終えます。

# SPRITE ON/OFF/STOP 〈ステートメント〉

## スプライトオン/オフ/ストップ

**機能** スプライトパターンが重なっておこる割り込みを許可、禁止、または保留します。

**書式** SPRITE ON
SPRITE OFF
SPRITE STOP

**文例** SPRITE ON

**解説** ON SPRITE GOSUB文で定義した割り込み処理ルーチンの制御を行ないます。スプライトパターンが表示されたとき、他のスプライトパターンと重なった場合に割り込みがかかります。

SPRITE ON 文は、スプライトが重なったときに割り込みを発生させ、割り込み処理ルーチンを実行します。

SPRITE OFF 文は割り込みを禁止します。

SPRITE STOP 文は割り込みを保留します。

SPRITE STOP文はSPRITE ONの実行を保留する命令であり、SPRITE ON状態でなければ割り込みは保留されずに無効となります。

**参照** ON SPRITE GOSUB, PUT SPRITE

# ON STOP GOSUB 〈ステートメント〉

オン ストップ ゴーサブ

**機能** CTRL ⟶ STOP キーによる割り込み処理ルーチンの開始行を定義します。

**書式** ON STOP GOSUB {行番号}

**文例** ON STOP GOSUB 500

**解説** STOP ON状態で、CTRL ⟶ STOP キーが押されたときに割り込みがかかり、プログラムの実行を中断して{行番号}から始まる割り込み処理ルーチンが実行されます。

割り込み処理ルーチンからの復帰はRETURN文を使います。このときRETURN文に行番号を指定していなければ割り込みがかかった際に、中断したプログラムが再開されます。

割り込み処理ルーチンを実行している間は、自動的にSTOP STOP状態になり、RETURN文実行とともにSTOP ONの状態になります。

この命令はプログラムモードのみ有効です。また、ON ERROR GOTO文、ON KEY GOTO文で定義する割り込み処理を実行中は、これらの処理ルーチンが終了するまで割り込みは保留されます。

ON STOP GOSUB文を使って、STOP ONの状態になったプログラムは中断することができなくなります。ON STOP GOSUB文は動作ミスのないプログラムにしてから使用してください。

**参照** ON KEY GOSUB, STOP ON/OFF/STOP

**サンプルプログラム：**

```
10 ON STOP GOSUB 40
20 STOP ON
30 GOTO 30
40 STOP OFF:PRINT"STOP!!"
50 IF INKEY$="" THEN 50
60 STOP ON:RETURN
```

CTRL ⟶ STOP キーを押すと"STOP!!"と表示されます。何かキーを押すことにより処理は再開します。

割込み

# STOP ON/OFF/STOP 〈ステートメント〉

ストップオン／オフ／ストップ

**機能** CTRL → STOP キーの割り込みを許可、禁止、保留します。

**書式** STOP ON
STOP OFF
STOP STOP

**文例** STOP ON

**解説** CTRL → STOP キーを押して発生する割り込みを制御します。
STOP ON文は割り込みの発生を許可します。CTRL → STOP キーが押されたときに、ON STOP GOSUB文で定義した割り込み処理ルーチンを実行します。
STOP OFF文は割り込みを禁止します。
STOP STOP文は、割り込みを保留します。
STOP STOP文は、STOP ONの実行を保留する命令であり、STOP ON状態でないときは、割り込みは保留されずに無効となります。
STOP STOP文により保留された割り込みは、次のSTOP ONが実行されたとき有効となり、ON STOP GOSUB文により割り込み処理ルーチンが実行されます。

**参照** ON STOP GOSUB



# ON STRIG GOSUB 〈ステートメント〉

オン エストリガー ゴーサブ

**機能** ジョイスティックまたはMX-101本体のトリガーボタンが押された場合にジャンプする割り込み処理ルーチンの開始行を定義します。

**書式** ON STRIG GOSUB〔{行番号}〕〔,{行番号}……〕

**文例** ON STRIG GOSUB 100, 300

**範囲** {行番号}は最大5個まで指定できる。

**解説** STRIG ON状態でスペースキーあるいはジョイスティックのトリガーボタンが押されたときに割り込みがかかり、プログラムの実行を中断して{行番号}から始まる割り込み処理ルーチンが実行されます。割り込み処理からの復帰は、RETURN 文を使います。
割り込み処理ルーチンを実行している間、自動的にSTRIG STOP状態になり、RE-TURN文実行とともに、STRIG ON状態になります。

**参照** STRIG ON/OFF/STOP

**サンプルプログラム：**

```
10 ON STRIG GOSUB 40,40
20 STRIG(1) ON
30 GOTO 30
40 PRINT "FIRE!":BEEP:RETURN
```

トリガーボタン1を押すと〝FIRE!〞が表示されます。

# STRIG ON/OFF/STOP 〈ステートメント〉

エストリガー オン／オフ／ストップ

**機能** ジョイスティックのトリガーボタンが押されたときに起こる割り込みを許可、禁止、保留します。

**書式** STRIG(|ジョイスティック番号|) ON
STRIG(|ジョイスティック番号|) OFF
STRIG(|ジョイスティック番号|) STOP

**文例** |ジョイスティック番号|は0～4、番号の指定はSTRIG関数と同じです。

**解説** トリガーボタンにより割り込み処理の制御を行ないます。
STRIG ON文は、プログラム実行中にトリガーボタンが押されたとき割り込みがかかり、ON STRIG ON GOSUB文で定義した割り込み処理ルーチンを実行します。
STRIG OFF文は、割り込みを禁止します。
STRIG STOP文は、割り込みを保留します。この状態で割り込みがかかったときにはSTRIG ON文を実行すると同時に割り込みが発生します。
ジョイスティックによってはトリガー2のボタンがないものもあります。

# DATA　〈ステートメント〉

データ

**機能**　READ 文で読み込む数値や文字定数を用意します。

**書式**　DATA {定数} 〔, {定数} ……〕

**文例**　DATA 154, 127, 234

**範囲**　定数は 1 行に入る範囲(255文字)の中で、カンマ(，)で区切ることもできます。

**解説**　DATA 文は READ 文で読み出す値を設定します。

プログラム実行時に DATA 文そのものは何も実行しません。

DATA 文は、ひとつのプログラム中にいくつも存在することができます。

DATA 文中の {定数} は、ひとつのプログラム内で連続されたデータの並びになります。

{定数} は、READ 文で読み出す型と同じ型が読み出さなければなりません。対応していないときは“Syntax error”となります。

DATA 文中の区切りは、カンマ( , )ですが、文字データとしてのカンマ( , )やピリオド( . )、意味をもった空白は、その文字定数の前後を引用符( " )で囲みます。

READ 文は行番号の小さい DATA 文に格納されているデータから順に読み込みますが、RESTORE 文では、READ 文で読み出す DATA 文を行単位で指定できます。

**参照**　READ, RESTORE

**サンプルプログラム**：

```
10 READ A$:IF A$="END" THEN 30
20 PRINT A$:GOTO 10
30 READ A:IF A=-1 THEN 50
40 PRINT A,:GOTO30
50 END
60 DATA "This is a Pen."
70 DATA "Is that a pen,too?"
80 DATA "No,it is not."
90 DATA END
100 DATA 10,2.5,&B1101,&O0265
110 DATA &HFC,-1
```

10行～20行では文字データのREAD文を実行しています。30行～40行は数値データのREAD文を実行します。

※数値データは、小数点を含んだ10進数、2進数、8進数、16進数が使えることを示しています。

# READ 〈ステートメント〉

リード

**機能** DATA 文で指定されているデータを読み込み、変数に代入します。

**書式** READ {変数リスト}

**文例** READ A, B, C$

**解説** DATA 文のなかのデータをひとつひとつ順に読み込み、指定した値を変数に代入します。変数を複数指定するときには、カンマ( , )で区切ります。

ひとつの READ 文で複数の DATA 文を参照したり、いくつかの READ 文でひとつの DATA 文を参照することができます。

READ 文で指定した変数名と DATA 文で読み出されるデータの型が一致していないときは、"Syntax error" になります。また変数の数が DATA 文のデータ数以上のときは "Out of DATA" エラーになります。

RESTORE 文で読み込む DATA 文を指定することができます。

**参照** RESTORE, DATA

**サンプルプログラム**：

```
10 RESTORE 60
20 READ A$,B,C
30 IF A$="END" THEN END
40 PRINT USING "&   & ### ### ###";A$;B;C;B+C
50 GOTO 20
60 DATA ABC,3,25
70 DATA DEFG,12,93
80 DATA HIJK,324,312
90 DATA END,0,0
```

READ文を使ってDATAを読み、表示します。"END" を読んだときにプログラムは終了します。

# RESTORE 〈ステートメント〉

リストア

**機能** READ 文で読み込むデータの開始行番号を指定します。

**書式** RESTORE〔{行番号}〕

**文例** RESTORE　700

**範囲** {行番号}は 0 ～65529

**解説** READ 文で読み出す DATA 文を指定しますが、行番号を省略すると、プログラム中の最初の DATA 文を指示したことになります。

**参照** DATA, READ

**サンプルプログラム**：

```
10 RESTORE 110 : GOSUB 30
20 RESTORE 70  : GOSUB 30 : END
30 READ A$,B,C
40 IF A$="END" THEN RETURN
50 PRINT USING"&   & ### ### ###";A$;B;C;B+C
60 GOTO 30
70 DATA ABC,3,24
80 DATA DEFG,123,93
90 DATA HIJK,324,312
100 DATA END,0,0
110 DATA ｱｲｳｴｵ,254,95
120 DATA ｶｷｸｹｺ,243,135
130 DATA END,0,0
```

RESTORE文を使って、自由な位置からDATAを読み込んでいます。

データ

# REM　〈ステートメント〉
## リマーク

**機能**　プログラムに注釈を入れます。

**書式**　REM〔{注釈文}〕

**文例**　REM＊＊＊PROGRAM＊＊＊
　　’　＊＊＊PROGRAM＊＊＊

**解説**　REM文以降を注釈行とします。
注釈行とは、プログラムを見やすくするために使うもので、自由にどこの行にでも置くことができます。(DATA文中には置けません。)
注釈文はプログラム実行時には無視されます。
REM文以降にコロン(：)でマルチステートメントにすることはできません。
REMの代りにアポストロフィ(’) SHIFT ⏎ 7 を使うことができます。

**サンプルプログラム：**

```
10 REM REMARK ノ レイ
20 ' REM ハ 「'」 デ ダイヨウデキマス。
30 PRINT "ギョウ ノ サイゴニモ ツケラレマス。" 'レイ
40 END
```

プログラムに注釈をつけた例です。

その他

# BASE 〈システム変数〉
ベース

**機能** 画面出力に関連しているテーブルのアドレスを与えます。

**書式** BASE（{数式}）

**文例** BASE（5）

**範囲** 0～19

**解説** 画面出力に関するアドレスを与えます。このシステム変数に値を代入することはできません。

{数式} で指定できるのは下記の通りです。

0 テキストモード（最大40×24文字）名称テーブル
1 テキストモード（最大40×24文字）使用しません
2 テキストモード（最大40×24文字）パターンジェネレータテーブル
3 テキストモード（最大40×24文字）使用しません
4 テキストモード（最大40×24文字）使用しません
5 テキストモード（最大32×24文字）名称テーブル
6 テキストモード（最大32×24文字）カラーテーブル
7 テキストモード（最大32×24文字）パターンジェネレータテーブル
8 テキストモード（最大32×24文字）スプライト属性テーブル
9 テキストモード（最大32×24文字）スプライトパターンテーブル
10 高解像度グラフィックモード　名称テーブル
11 高解像度グラフィックモード　カラーテーブル
12 高解像度グラフィックモード　パターンジェネレータテーブル
13 高解像度グラフィックモード　スプライト属性テーブル
14 高解像度グラフィックモード　スプライトパターンテーブル
15 マルチカラーモード　名称テーブル
16 マルチカラーモード　使用しません
17 マルチカラーモード　パターンジェネレータテーブル
18 マルチカラーモード　スプライト属性テーブル
19 マルチカラーモード　スプライトパターンテーブル

**サンプルプログラム：**

```
10 SCREEN 0
20 A=BASE(0)
30 FOR J=0 TO 255
40   VPOKE A+J,J
50 NEXT J
60 LOCATE 0,10
70 END
```

スクリーンモード0の名称テーブルの先頭を求め、VRAMに直接データを書き込みます。

# VPEEK 〈関数〉
ブイピーク

**機能** VRAM上の指定された番地の内容を読み出します。

**書式** VPEEK（{番地}）

**文例** A = VPEEK(1) [その結果] Aには VRAMの1番地の内容が入る。

**範囲** {番地}は0～16383（&H0～&H3FFF）

**解説** VRAM上にある画面上のデータを読みます。読み出されるデータは0～255の範囲です。

**参照** VPOKE, BASE

**サンプルプログラム：**

```
10 A$=""
20 FOR J=0 TO 39
30  A$=A$+CHR$(VPEEK(J))
40 NEXT J
50 PRINT A$
60 END
```

画面の一番上の行を直接VRAMから読み、A$に入れ表示します。



# VPOKE 〈ステートメント〉
ブイポーク

**機能** VRAM上の指定された番地にデータを書き込みます。

**書式** VPOKE {番地}, {データ}

**文例** VPOKE A%, &HFF

**範囲** {番地}はVRAMの先頭番地を0とする0～16383 {データ}は0～255

**解説** VRAM上に直接データを書き込みます。

**参照** VPEEK, BASE

（注）不用意に使用すると画面表示が変化する場合があるのでVDPを理解したうえで使用してください。

**サンプルプログラム：**

```
10 SCREEN 0
20 FOR J=0 TO 39
30 VPOKE J,INT(RND(1)*256)
40 NEXT J
50 END
```

VRAM上に直接データを書きます。文字は一番上の行に表示されます。

# CLS　〈ステートメント〉

## クリアースクリーン

**機能**　画面を消去します。

**書式**　CLS

**文例**　CLS

**解説**　画面上の内容をすべて消します。PRINT CHR$(12); も同じ動作をします。
カーソルの位置は(0, 0)となります。LP (最終参照点) は変化しません。ファンクションキーの表示指定(KEY ON 状態)のときは、ファンクションキーの表示はのこります。
グラフィックモードのとき、背景色は COLOR 文で指定した色に塗られます。LP は実行前と同じです。

**サンプルプログラム：**

```
10 FOR J=0 TO 100
20   X=INT(RND(1)*32)
30   Y=INT(RND(1)*20)
40   LOCATE X,Y:PRINT "*";
50 NEXT J
60 FOR J=0 TO 1000:NEXT J
70 LOCATE 8,17:PRINT "              "
80 LOCATE 8,18:PRINT "   CLS シマス   "
90 LOCATE 8,19:PRINT "              "
100 FOR J=0 TO 500:NEXT J
110 CLS
120 END
```

10行から100行で画面に“＊”を表示します。その後に時間をおいて、CLS（クリアースクリーン）命令を実行し、画面を消します。70行と90行の空白(スペース)は、14です。

# COLOR　〈ステートメント〉

カラー

**機能**　画面の各部の色を指定します。

**書式**　COLOR〔{前景色}〕〔,{背景色}〕〔,{周辺色}〕

**文例**　COLOR 15, 6, 3

**範囲**　色コードは0～15

**解説**　画面の各部分の色を指定します。

前景色はテキスト画面のときの文字の色や、グラフィック画面の点や線の色です。背景色は画面の背景の色、周辺色は画面のわくになる色です。SCREEN 0では、周辺色は背景色と同じになるので、周辺色を指定しても意味を持ちません。

起動時は、COLOR 15, 4, 7になっています。

前景色は文字表示のほか、PSET, LINE, CIRCLE 命令のときに色指定を省略したときの値になります。また、背景色はPRESET 命令の省略値になります。

カラーコード表

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 透　明 | 4 | 暗い青 | 8 | 赤 | 12 | 暗い緑 |
| 1 | 黒 | 5 | 明るい青 | 9 | 明るい赤 | 13 | 紫 |
| 2 | 緑 | 6 | 暗い赤 | 10 | 黄 | 14 | 灰 |
| 3 | 明るい緑 | 7 | 水　色 | 11 | 明るい黄色 | 15 | 白 |

**サンプルプログラム：**

```
10 SCREEN 1
20 CLS
30 FOR J=0 TO 100
40   X=INT(RND(1)*32)
50   Y=INT(RND(1)*20)
60   C=INT(RND(1)*96)+32
70   LOCATE X,Y:PRINT CHR$(C);
80 NEXT J
90 FOR J=0 TO 300:NEXT J
100 FOR J=1 TO 50
110 F=INT(RND(1)*16)
120 B=INT(RND(1)*16)
130 G=INT(RND(1)*16)
140 COLOR F,B,G
150 FOR K=1 TO 300:NEXT K
160 NEXT J
170 COLOR 15,4,7
180 LOCATE 0,20
190 END
```

画面をスクリーン1にし、101個の適当な文字を画面に表示し、前景色、背景色、周辺色を適当に変化させます。

画面制御

# VDP
ブイデーピー

〈システム変数〉

**機能** VDP（ビデオディスプレイプロセッサ）のレジスタへデータを書き込んだり参照したりします。

**書式** VDP（{レジスタ番号}）

**文例** X = VDP(8)

**範囲** {レジスタ番号}は0～8

**解説** VDPレジスタの内容を参照したり変更したりします。
{レジスタ番号}は次のように指定します。

0：各種機能、表示モード
1：各種機能、表示モード
2：パターン名称テーブル・ベースアドレス
3：カラーテーブル・ベースアドレス
4：パターンジェネレータ・ベースアドレス
5：スプライト属性テーブル・ベースアドレス
6：スプライトジェネレータ・テーブル・ベースアドレス
7：テキストカラー、バックドロップカラー
8：割り込みフラグ/第5スプライト情報/衝突フラグ(読み出し専用)

注）このシステム変数は、VDPを理解した上で使用してください。

# PRINT　　〈ステートメント〉
プリント

**機能**　画面にデータを表示します。

**書式**　PRINT〔{式}……〕

**文例**　PRINT "XYZ"

**解説**　{式}で指定された数値や文字列を画面に表示します。{式}が省略された場合は改行だけ行ないます。

複数個の{式}を指定する場合は、式の間をカンマ(,)かセミコロン(;)で区切ります。その使いわけは次の通りです。

カンマ＝区切られた次の領域の始めから表示され、頭そろえを行なうことができます。

セミコロン＝区切られた直前にプリントしたもののすぐうしろに続いて{式}の内容が表示されます。なお、文字式の区切り記号として空白(" ")を使うと、これと同じ働きをします。

一番最後の{式}のうしろにセミコロンがあれば、次に実行される PRINT 文で出力する{式}の内容がすぐあとに表示されます。

変数と" "で囲まれた文字列との区切りは、区切り記号(,)(;)を省略することができます。このときセミコロンを使用した状態になります。

数値を表示した場合は、そのうしろに必ず空白がひとつ表示されます。

数値の前には符号を表示するため、正の値ならば空白、負の値ならばマイナス符号が表示されます。

PRINT の簡略形として疑問符(?)を使用することができます。

なお、PRINT 文はテキストモードでのみ有効です。

**サンプルプログラム：**

```
10 FOR J=1 TO 10
20  PRINT "ABCD";
30 NEXT J: PRINT
40 FOR J=1 TO 10
50  PRINT "EFGH",
60 NEXT J
70 END
```

"ABCD"の文字を続けて10回表示します。30行のPRINT文で改行して"EFGH"を空白をつけて10回表示します。

# PRINT USING 〈ステートメント〉

## プリントユージング

**機能** 文字列や数値を指定された書式で表示します。

**書式** PRINT USING {書式}；{式}〔{；|，}{式}{；|，}……〕

**文例** PRINT USING "¥¥##"；X，Y

**解説** {式}で表わされる文字列や数値を{書式}の型にしたがって編集し、テキスト画面に表示します。この{式}は、カンマ( , )やセミコロン( ; )で区切れば、複数個でも指定できます。

{書式}は、次にあげる書式制御文字を連ねた文字列であり、文字列を編集するものと数値を編集するものとがあります。

**●文字列を編集する書式制御文字**

！………文字列の左側１字を表示します。

(例)

```
P$="American" ↵
Ok
PRINT USING"!";P$ ↵
A
Ok
```

&{n個の空白}&……

文字列の左側から（n＋2）文字を表示します。文字列が（n＋2）文字より長い場合は余分な文字を表示しません。また（n＋2）文字より短かければ足らない文字数だけの空白を右側に補って表示します。

```
P$ = "American" ↵
Ok
PRINT USING "&□□□&" ;P$ ↵
Ameri
Ok
```

@……{書式}の文字列のなかに含まれている"@"を{式}の文字列でおき換え、{書式}を表示します。ひとつの"@"は左側の{式}から順に対応し、{書式}中の"@"の数が{式}の数より多い場合は残りの"@"は無視します。

(例)

```
P$="He":Q$="American" ↵
Ok
PRINT USING"@ is @" ;P$,Q$ ↵
He is American
Ok
```

テキスト画面への出力

●**数値を編集する書式制御文字**

#……数値を"#"で指定された桁数だけ表示します。数値が指定された桁数より小さい場合は右づめで表示し、左側に空白を補います。

.……小数点の位置を指定し、小数部の桁数が書式で指定された桁数より小さい場合は、右側に0を補います。

(例)

```
PRINT USING "###.##";3.5,4.792,7,.92 ↵
 3.50  4.79  7.00  0.92
Ok
```

+……{書式}の左端に"+"が付くと、数値の前に正負符号を表示し、{書式}の右端に"+"が付くと数値のうしろに正負符号を表示します。
また"+"が2個以上並んだ場合は、数値の外側の"+"は制御文字とみなしません。

(例)

```
PRINT USING "+##.##";2.34,-2.34 ↵
 +2.34    -2.34
Ok
PRINT USING"##.##+";2.34,-2.34 ↵
 2.34+   2.34-
Ok
```

−……{書式}の右端に"−"が付くと、数値が負の場合は数値のうしろに"−"を表示し、{書式}の左端に付いたり、2個以上並んだときは制御文字とみなしません。

(例)

```
PRINT USING"##.##-";2.34,-2.34 ↵
 2.34   2.34-
Ok
```

**…{書式}の左端に"##"の代わりに"**"がくると、数値の整数部の桁数が、指定された桁数より小さい場合、左側に"*"を補います。

(例)

```
PRINT USING"**##.#";2.34,-2.34 ↵
***2.3  **-2.3
Ok
```

¥¥…{書式}の左端に"¥¥"が付くと、数値の前に"¥"を表示します。"¥¥"は"##"の指定と同様に、2桁分の表示桁を確保しますが、"¥"を表示するために、このうちの1桁を使います。"¥¥"は指数形式の書式指定をしているときは使えません。

(例)

```
PRINT USING "¥¥##.##";23.45,-23.45 ↵
 ¥23.45   -¥23.45
Ok
```

```
PRINT USING"¥¥##.##-";23.45,-23.45 ↵
 ¥23.45   ¥23.45-
Ok
```

**＊＊¥**‥{書式}の左端に〝＊＊¥〞が付くと、数値の前に¥を表示します。数値の桁数が指定された桁数より小さい場合、左側に〝＊〞を補います。
〝＊＊¥〞は〝＊＊＊〞の指定と同様3桁分の表示桁を確保しますが、〝¥〞を表示するために、このうちの1桁を使います。〝＊＊¥〞は指数形式の書式指定をしているときは使えません。

(例)

```
PRINT USING"**¥#.##";23.45 ↵
 *¥23.45
Ok
```

**,**……〝,〞が{書式}の整数部の〝#〞の間に入った場合、数値の整数部を3桁ごとに〝,〞で区切って表示します。〝,〞が小数部(小数点より右側)に入った場合は数値の右側に〝,〞を表示し、3桁ごとの区切りは行ないません。〝,〞は指数形式の書式指定しているときは使えません。

(例)

```
PRINT USING"####,.##";2345.6 ↵
 2,345.60
Ok
PRINT USING"####.##,";2345.6 ↵
 2345.60,
Ok
```

**^^^^**…〝^^^^〞が桁数指定の〝#〞のあとに付くと、数値を指数形式で表示します。
また、{書式}の先頭に〝+〞か{書式}のあとに〝+〞や〝-〞を指定していない場合、数値が正ならば数値の前に1桁の空白が、負ならば〝-〞が表示されます。

(例)

```
PRINT USING"##.##^^^^";234.56 ↵
 2.35E+02
Ok
PRINT USING"#.##^^^^-";-23.45 ↵
 2.35E+01-
Ok
PRINT USING"+##.##^^^^";23.45,-23.45 ↵
 +2.35E+02-2.35E+01
Ok
```

**%**……表示しようとする編集済みの数値が、書式で指定された桁数より大きいとき、数値の直前に〝%〞を表示します。また、表示しようとする数値が四捨五入によって{書式}で指定された桁数より大きくなった場合も同様に表示します。

(例)

```
PRINT USING".###";.9999⏎
 %1.000
Ok
PRINT USING"##.##";234.56⏎
 %234.56
Ok
```

{書式}中に以上の書式制御文字以外の文字がある場合、文字の位置によって数値の前やうしろにその文字を表示します。

(例)

```
PRINT USING"TOTAL**¥####,";2000⏎
TOTAL**¥2,000
Ok
```

**サンプルプログラム**：

```
10 PRINT USING"¥¥###";45
20 PRINT USING"+####";654
30 PRINT USING"&      &";"UHURIEBFHDFS"
40 END
```

PRINT USING文を使っていろいろな形で表示します。

## LOCATE 〈ステートメント〉

ロケート

**機能** テキスト画面のカーソルの位置を指定します。

**書式** LOCATE〔{X}〕〔, {Y}〕〔, {カーソルスイッチ}〕

**文例** LOCATE 0, 10

**範囲** X, Yはテキスト画面の座標範囲、たとえばSCREEN 0ではXは0～39, Yは0～23です。

**解説** カーソルをテキスト画面のキャラクター座標(X, Y)の位置に移動します。
{カーソルスイッチ}はカーソルの状態を変更します。
カーソルスイッチ　0：カーソルは入力待ち以外は表示しない。(省略時は0)
カーソルスイッチ　1：カーソルを表示させる。
初期値は0です。

**サンプルプログラム**：

```
10 LOCATE INT(RND(1)*40),INT(RND(1)*20
20 PRINT "X";
30 GOTO 10
```

LOCATE文で画面上の適当な所に文字を表示するプログラムです。CTRL ＋ STOP で終了します。

# POS 〈システム変数〉
ポジション

**機能** 画面上のカーソルの桁位置（水平方向）を与えます。

**書式** POS（{式}）

**文例** X ＝ POS(0)

**範囲** {式}はダミーの引数

**解説** キャラクター座標の左から水平位置を与えます。
この関数はテキストモードでのみ有効です。
式は通常0を使います。

**参照** CSRLIN

**サンプルプログラム：**

```
10 SCREEN 0:WIDTH 40
20 LOCATE INT(RND(1)*40),INT(RND(1)*23)
30 X=POS(0):Y=CSRLIN
40 PRINT"$";
50 LOCATE 10,24
60 PRINT USING"X:## Y:##";X;Y;
70 IF INKEY$="" THEN 70
80 GOTO 10
```

乱数で表示した＄の位置をCSRLIN, POS(0)を使って求めています。

# TAB 〈関数〉
タブ

**機能** 指定された桁まで空白を与えます。

**書式** TAB（{数式}）

**文例** PRINT〝CASIO〞；TAB(10)；〝PV−16〞

**範囲** {数式}の値は0〜255

**解説** TABはPRINT文、LPRINT文で用いることができます。
LETなどの代入文では使用できません。
TABは指定した桁から文字列や数値を表示するときに使います。
{数式}の値が適切でないときは“Illegal function call”エラーになります。

**参照** SPC

**サンプルプログラム：**

```
10 FOR J=1 TO 10
20 PRINT TAB(J);"ABC"
30 NEXT J
40 END
```

TAB関数を利用して〝ABC〞を10個斜めに並べて表示します。

## WIDTH　〈ステートメント〉

ウィドス

**機能**　画面に表示する文字の桁数を指定します。

**書式**　WIDTH {桁数}

**文例**　WIDTH 20 [その結果] 画面は20桁表示に指定される。

**範囲**　SCREEN 0 (最大40×24文字)のときは1～40、SCREEN 1 (最大32×24文字)のときは1～32

**解説**　画面上に表示する1行の桁数を設定します。
初期値はSCREEN 0で39桁、SCREEN 1で29桁となっています。

**参照**　SCREEN

**サンプルプログラム**

```
10 SCREEN 0
20 WIDTH 20
30 FOR J=1 TO 40 : PRINT J; : NEXT J
40 WIDTH 40
50 FOR J=1 TO 40 : PRINT J; : NEXT J
60 END
```

WIDTH文で1行の表示桁をかえて文字を表示します。

## CSRLIN　〈システム変数〉

カーソルライン

**機能**　現在のカーソルの行位置を与えます。

**書式**　CSRLIN

**文例**　X = CSRLIN

**範囲**　結果は0～23です。

**解説**　現在のカーソルの行位置を調べます。
画面の最上行が0、最下行が23です。
CSRLINのようなシステム変数に値を代入することはできません。
カーソルの桁位置(水平方向)を知るためには、POS 関数があります。

**参照**　POS, LOCATE

**サンプルプログラム**：

```
10 LOCATE 10,3
20 PRINT "TEST";
30 PRINT "PRINT"
40 LOCATE 10,5
50 PRINT "TEST"
60 Y=CSRLIN-1
70 LOCATE 10,15
80 PRINT "ESCAPE"
90 LOCATE 14,Y
100 PRINT "PRINT"
110 END
```

画面上に"TEST"と"PRINT"という表示をしています。20行～30行は連続して表示されますが、50行～100行では"TEST"を表示した後に"ESCAPE"を異なる位置に表示し、"PRINT"を再び"TEST"の後に表示します。

テキスト画面への出力

# CIRCLE　〈ステートメント〉
## サークル

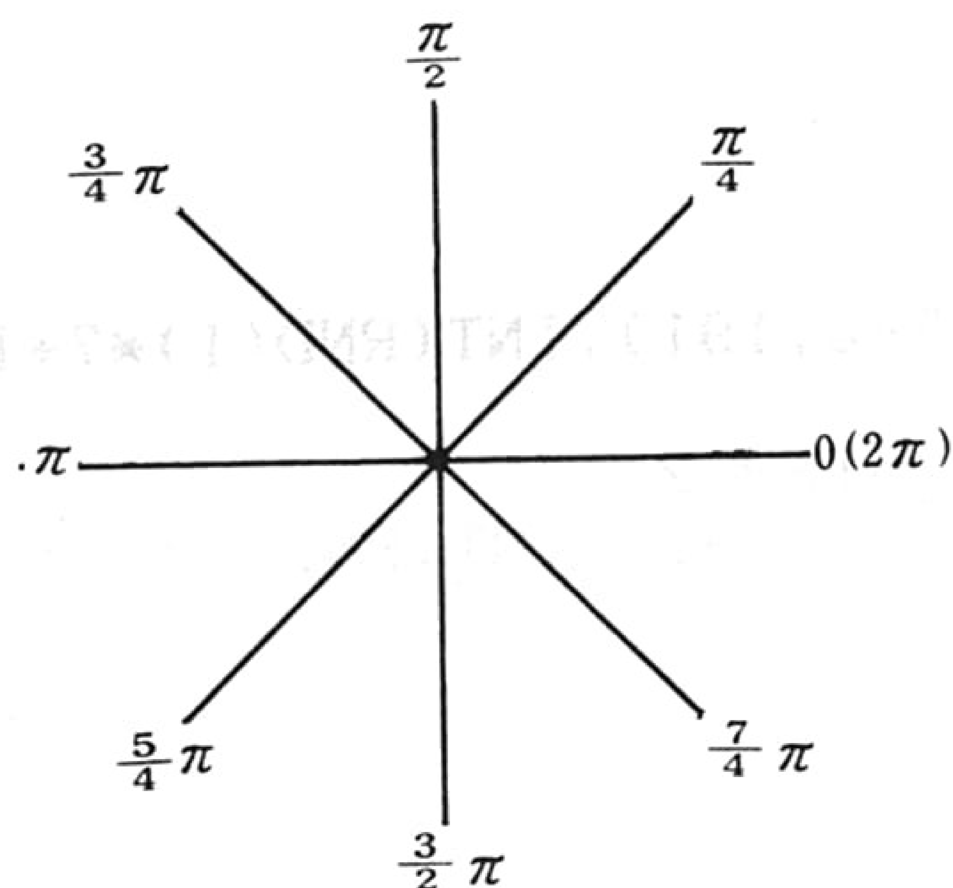
**機能**　グラフィックモードの画面上に、円または楕円を描きます。

**書式**　CIRCLE |(X, Y) / STEP (X, Y)| ,{半径}〔,{カラーコード}〕〔,{開始角度}〕〔,{終了角度}〕〔,{縦横比率}〕

**文例**　CIRCLE ( 0, 191), 100, 5

**範囲**　X, Y, STEP (X, Y) の座標はX座標の値が0～255, Y座標が0～191以内
カラーコードは0～15
{開始角度}、{終了角度}は、$-2\pi \sim +2\pi$
範囲を越えると"Illegal function call"エラーになります。

**解説**　(X, Y) を中心とし、{半径}で指定する大きさの円を描きます。{カラーコード}を指定すると、指定した色で円を描きますが、省略された場合は、COLOR文で指定している前景色になります。
また、{開始角度}、{終了角度}を指定すると、指定角度の範囲のみ円弧を書くことができます。
角度の指定の単位はラジアンです。
省略値は0と$2\pi$になります。




$\pi = 3.1415926535898$

{縦横比率}は(垂直方向の半径)/(水平方向の半径)で指定されます。{縦横比率}を省略したときは、1となります。
{縦横比率}が1以下の場合には{半径}は水平方向の半径を意味し、1以上のときは、垂直方向になります。
中心座標は、STEPを付けて最終参照点(LP)からの相対座標で指定することも可能です。
CIRCLE命令後のLPは中心座標(X, Y)に設定されます。

**参照**　COLOR, SCREEN

**サンプルプログラム：**

```
10 SCREEN 2
20 CIRCLE (128,96),50,10,,,1
30 CIRCLE (128,96),50,10,,,2
40 CIRCLE (128,96),50,10,,,3
50 FOR J=0 TO 2000:NEXT J
60 END
```

縦横比率をかえた3つの円を描きます。

グラフィック画面への出力

# LINE

ライン

〈ステートメント〉

**機能** グラフィックモードの画面上に直線や四角形を描きます。

**書式**

LINE [ {($W_{X1}$, $W_{Y1}$) | STEP ($X_1$, $Y_1$)} ] − {($W_{X2}$, $W_{Y2}$) | STEP ($X_2$, $Y_2$)} [, {カラーコード}] [, {B | BF}]

**文例** LINE (50 , 50) − (100 , 50), 8

**範囲** グラフィックコマンドであり、SCREEN 2またはSCREEN 3で実行します。$W_{X1}$, $W_{X2}$, $W_{Y1}$, $W_{Y2}$は、すべてグラフィック座標のとることのできる範囲（X座標は0～255、Y座標は0～191）になります。STEP ($X_1$, $Y_1$)、STEP ($X_2$, $Y_2$) の値もグラフィック座標の値になります。

**解説** 指定した2点（$X_1$, $Y_1$）と（$X_2$, $Y_2$）を結ぶ線をカラーコードで引きます。カラーコードが省略されたときは、COLOR文で定義してある前景色になります。始点を省略した場合は最終参照点（LP）からになります。

命令が実行されると，LP ($X_2$, $Y_2$) の値になります。STEPを付けた場合はLPからの相対座標になります。

Bを付けると2点間を対角線とする四角形を描き、BFをつけると2点間を対角線とする四角形を描いた上で、さらにその四角形の中を塗りつぶします。

**サンプルプログラム：**

```
10 SCREEN 2
20 DEFINT A-Z
30 FOR J=0 TO 255
40  LINE(J,0)-(255-J,191),INT(RND(1)*7+1)
50 NEXT J
60 FOR J=0 TO 125 STEP 2
70  LINE(J,0)-(J+128,191),INT(RND(1)*7+1),B
80 NEXT J
90 GOTO 90
```

30行～50行で画面いっぱいに線を引いています。
60～80行では、四角形を描いています。

# DRAW　〈ステートメント〉

## ドロー

**機能**　画面に図形を描きます。

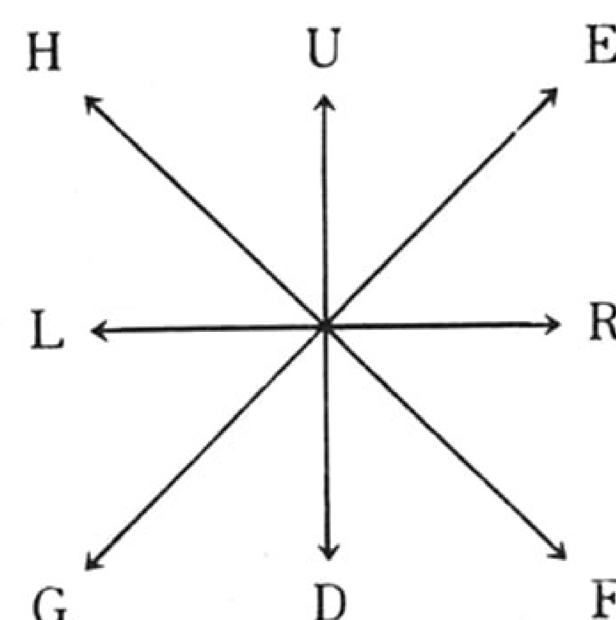
**書式**　DRAW {文字式}

**文例**　DRAW "BM10，10D40R70U40L70"

**解説**　画面にある点(参照点)を移動させながら線を描きます。DRAW文はグラフィックモードでのみ使うことができます。
{文字式}には、参照点の位置設定、移動方向と距離、角度などを指定するグラフィックマクロ命令をひとつ以上連続して記述します。ここで使用するグラフィックマクロ命令には、移動、回転、色指定、スケールファクター、ローカルマクロ命令の5種類があります。
次にその5種類について説明します。

●移動マクロ命令

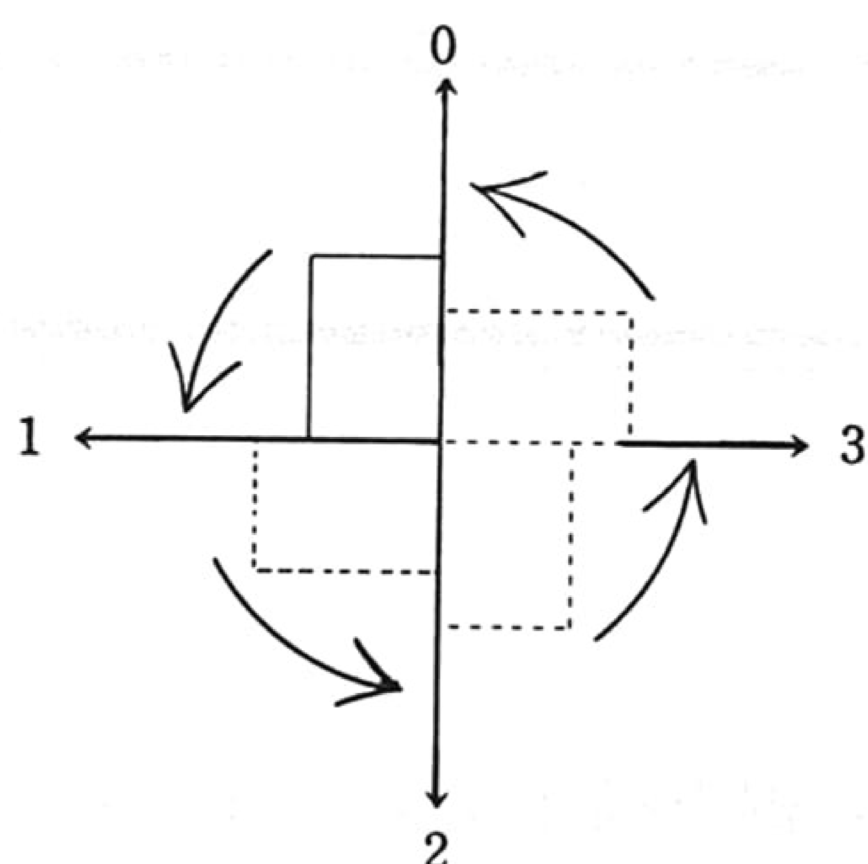
| | |
|---|---|
| U {距離}…… | 上へ移動 |
| D {距離}…… | 下へ移動 |
| L {距離}…… | 左へ移動 |
| R {距離}…… | 右へ移動 |
| E {距離}…… | 右斜め上へ移動 |
| F {距離}…… | 右斜め下へ移動 |
| G {距離}…… | 左斜め下へ移動 |
| H {距離}…… | 左斜め上へ移動 |
| $M_{X,Y}$………… | 移動位置をX方向、Y方向各々についての絶対座標あるいは、相対座標で指定します。ここでXの前にプラス(+)またはマイナス(−)の符号が書かれていれば、参照点からの相対座標指定と解釈されます。 |
| B………… | 各々の方向へ描画をせずに移動します。 |
| N………… | 各々の方向へ描画しながら移動しますが、描画後の参照点は始点となります。 |




上の各コマンド中の{距離}は、各方向への移動距離を画素(ドット)単位で指定しますが、実際の移動距離は、これにSコマンドで設定されるスケールファクターを乗じたものとなります。また、各コマンドに同じ{距離}を指定しても、画面上での実際の移動距離は、図のように斜めの場合と、縦、横方向ではそれぞれ異なっています。
また、B、Nのマクロ命令はU～Mのマクロ命令の直前に付加して使用します。

●回転マクロ命令

A {角度}　U、D、L、R、E、F、G、Hおよび相対座標指定のときのMマクロで描画する図形を90°単位で回転して表示します。これは一度指定すると次のAマクロまで有効です。
{角度}は0～3の数字で表し、0は0度、1は90度、2は180度、3は270度を表します。




●色指定マクロ命令

C｛カラーコード｝…｛カラーコード｝で指定した色で図形を描きます。｛カラーコード｝は0～15までです。（186P参照）

●スケールファクターマクロ命令

S｛整数｝…スケールファクターを設定します。
｛整数｝に指定できる値は1～255の範囲で、この値の4分の1がスケールファクターとなります。(たとえば｛整数｝に1を指定した場合、スケールファクターは1/4となります。) U、D、L、R、E、F、G、HマクロおよびMマクロでの実際の移動距離は、距離にスケールファクターを乗じたものとなります。また｛整数｝の指定を省略した場合は0が設定されます。ただし0は4と同じスケールファクターになります。Sマクロを設定すると、次のSマクロまで有効です。

●ローカルマクロ命令

X｛文字変数｝;…………………｛文字変数｝中の文字列に含まれているグラフィックマクロ命令を実行します。
｛文字変数｝には上記のグラフィックマクロをあらかじめ入れておきますが、このとき｛文字変数｝のあとのセミコロン(;)は必ず入れておきます。ローカルマクロはグラフィックマクロの文字列が255を越える場合や繰り返し図形を描くときなどに便利です。

グラフィックマクロ命令では｛距離｝や｛角度｝などのパラメータに定数を指定しますが〝=｛変数名｝;〃の形式にすれば、数値変数を指定することができます。この場合｛変数名｝のうしろにはセミコロン(;)を必ず入れます。文字式に誤りがあると“Illegal function call”エラーになります。

**サンプルプログラム：**

```
10 COLOR 9,1,1
20 SCREEN 2
30 DEFINT A-Z
40 D=RND(-TIME)
50 DEFFNR(X)=RND(1)*X
60 D$="C9BD5E5F5D10L10U10F10BL10E10L10"
70 FOR J=1 TO 100
80 PRESET(FNR(256-10),FNR(192-15))
90 DRAW D$
100 NEXT J
```

60行のD$で家の形を定義します。その家を70行～100行で画面上に表示します。

## PSET 〈ステートメント〉
### ピーセット

**機能** 指定された座標上に点を描きます。

**書式** PSET |(X, Y) / STEP(X, Y)| 〔, {カラーコード}〕

**文例** PSET(50, 100), 7

**範囲** 座標の範囲はX座標が0～255、Y座標が0～191のとき画面上に表示されます。
{カラーコード}は0～15

**解説** 指定した座標に{カラーコード}で示した色の点を打つ。STEPをつけた場合、最終参照点(LP)からの相対座標となります。
{カラーコード}を省略したときは、COLOR文で指定した前景色となります。PSET命令の実行後、LPはPSET命令で指定した値になります。
またこの命令はグラフィックモード(SCREEN 2, SCREEN 3)でのみ有効で、テキストモードでは"Illegal function call" エラーとなります。

**参照** COLOR, PRESET

## PRESET 〈ステートメント〉
### ピーリセット

**機能** 指定された座標上に描かれた点を消します。

**書式** PRESET |(X, Y) / STEP(X, Y)| 〔, {カラーコード}〕

**文例** PRESET (30, 50)

**範囲** X, Y 座標

**解説** (X, Y) で表わされる位置に描かれた点を消します。
STEP を付けた場合は最終参照点 (LP) からの相対座標による指定となります。
{カラーコード}は点を消すときに使う色を指定します。これを省略した場合は現在の背景色になります。
PRESET はグラフィックモード(SCREEN 2 またはSCREEN 3)でだけ使うことができます。
PRESET 文を実行した結果、LP は指定した座標へ移動します。

**参照** PSET, COLOR

**サンプルプログラム：**

```
10 SCREEN 2
20 LINE(0,0)-(255,191),1,BF
30 DEF FNC(X)=INT(RND(1)*X)
40 PSET(FNC(256),FNC(192)),FNC(7)+1
50 PRESET (FNC(256),FNC(192))
60 GOTO 40
```

画面上に点を点滅させます。

グラフィック画面への出力

# PAINT 〈ステートメント〉
ペイント

**機能** 指定された境界色で囲まれた領域を指定された色で塗りつぶします。

**書式** PAINT ｜(X, Y) / STEP(X, Y)｜〔, {領域色}〕〔, {境界色}〕

**文例** PAINT (50, 50) 4, 7

SCREEN 2, または SCREEN 3 のグラフィックモードで使用します。

**範囲** 座標の値はX座標は0～255、Y座標は0～191、領域色・境界色は0～15

**解説** (X, Y) で指定する座標を含む{境界色}で囲まれた領域を、指定された{領域色}で塗りつぶします。

STEP を付けると最終参照点(LP)からの相対座標となります。

{領域色}、{境界色}はともにカラーコードで指定します。{領域色}を省略した場合、COLOR 文で指定した前景色が用いられます。{境界色}を省略した場合には{領域色}の指定と同じカラーコードが{境界色}として用いられます。

(注) {境界色}はマルチカラーモード SCREEN 3 でのみ使用することが可能で、高解像度グラフィックモード SCREEN 2 では使うことができません。この場合、{領域色}と同じ色で囲まれた領域が塗りつぶされます。

(X, Y) は塗り始める座標を指定します。もしこの点が、すでに指定された境界色と同じ色であった場合には、PAINT は塗りつぶしを行ないません。

PAINT 文はグラフィックモードでのみ使うことができます。

**参照** SCREEN, COLOR

**サンプルプログラム：**

```
10 SCREEN 2
20 LINE(10,10)-(200,100),4,B
30 LINE(50,50)-(250,190),4,B
40 PAINT(20,20),1,4
50 PAINT(60,60),2,4
60 PAINT(210,110),3,4
70 GOTO 70
```

高解像度モードにして塗りつぶされた四角形を描きます。その後、色コードの1、2、3でPAINTをしますが、高解像度モードのため境界色が意味をもちませんので、画面全体の色が変ってしまいます。

# POINT 〈関数〉
ポイント

**機能** 指定された座標のドットの色を与えます。

**書式** POINT (X, Y)

**文例** A = POINT (50, 50) その結果 Aには座標 (50, 50) のドットカラーコードが入る。

**解説** (X, Y) で指定した座標上に表示されているドットのカラーコードを与えます。

POINT 関数はグラフィックモードでのみ有効です。

スプライトパターンの色を調べることはできません。

POINT 関数を指定しても、LP は変わりません。

**サンプルプログラム：**

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
30 FOR J=0 TO 24
40  PRESET(30,J*8)
50  COLOR INT(RND(1)*15)+1
60  PRINT #1,"◆"
70 NEXT J
80 '
90 FOR J=0 TO 24
100  C=POINT(34,J*8+4)
110  PRESET(45,J*8)
120  COLOR 15:PRINT #1,"COLOR=";C;
130 NEXT J
140 GOTO 140
```

いろいろな色で◆を表示し、その色をPOINT関数で求めて表示します。

# SPRITE$ 〈システム変数〉

## スプライトダラー

**機能** スプライトパターンを設定します。

**書式** SPRITE$({スプライトパターン番号})=文字式

**文例** SPRITE$(0)=P$

**範囲** {スプライトパターン番号}は、スプライトサイズが8×8ドットのときは0～255 16×16ドットのときは0～63。

**解説** スプライト表示をするためのスプライトパターンの設定を行ないます。

①8×8ドットの定義方法

8×8ドットのパターンの定義は横8ドットを1文字(1バイト)とし、8文字分の文字列として与えます。点がある部分を1、ない部分を0とし、2進数で表わすことができます。

**8×8ドットのスプライトパターン**

①CHR$(&B00010000)
②CHR$(&B00111000)
③CHR$(&B00111000)
④CHR$(&B00111000)
⑤CHR$(&B01111100)
⑥CHR$(&B11111110)
⑦CHR$(&B11111110)
⑧CHR$(&B00101000)

横8ドットを1組として、それぞれを1バイトのビット列(0と1の列)で表わします。点を描く部分が1で、点を描かない部分は0です。(計8バイト)

②16×16ドットの定義方法

8×8ドットのときと同じように横8ドットを1文字（1バイト）として定義します。16×16ドットの場合を表わすには8×8ドットの区画に4分割して、1区画8文字を下図の順番に並べて合計32の文字列になります。

**16×16ドットのスプライトパターン**

| | |
|---|---|
| 1 | 3 |
| 2 | 4 |

①CHR$(&B00000001)
②CHR$(&B00000001)
③CHR$(&B00001111)
④CHR$(&B00011111)
⑤CHR$(&B00011101)
⑥CHR$(&B00011101)
⑦CHR$(&B00011111)
⑧CHR$(&B11111111)
⑨CHR$(&B11111110)
⑩CHR$(&B01111100)
⑪CHR$(&B01111110)
⑫CHR$(&B00111111)
⑬CHR$(&B00001111)
⑭CHR$(&B00000101)
⑮CHR$(&B00000100)
⑯CHR$(&B00111110)
⑰CHR$(&B10000000)
⑱CHR$(&B10000000)
⑲CHR$(&B11110000)
⑳CHR$(&B11111000)
㉑CHR$(&B10111000)
㉒CHR$(&B10111000)
㉓CHR$(&B11111000)
㉔CHR$(&B11111111)
㉕CHR$(&B01111111)
㉖CHR$(&B00111110)
㉗CHR$(&B01111110)
㉘CHR$(&B11111100)
㉙CHR$(&B11110000)
㉚CHR$(&B10100000)
㉛CHR$(&B00100000)
㉜CHR$(&B01111100)
(計32バイト)

参照　PUT SPRITE

**サンプルプログラム：**

```
10 SCREEN 2,0
20 SPRITE$(0)=STRING$(8,255)
30 PUT SPRITE 0,(100,100),15
40 IF INKEY$="" THEN 40
50 SCREEN 2,1
60 SPRITE$(0)=STRING$(8,255)
70 PUT SPRITE 0,(100,100),15
80 IF INKEY$="" THEN 80
90 SCREEN 2,2
100 SPRITE$(0)=STRING$(16,255)
110 PUT SPRITE 0,(100,100),15
120 IF INKEY$="" THEN 120
130 SCREEN 2,3
140 SPRITE$(0)=STRING$(16,255)
150 PUT SPRITE 0,(100,100),15
160 IF INKEY$="" THEN 160
170 END
```

各種のスプライトサイズを表示します。キー入力があるごとにスプライトサイズが変化します。

# PUT SPRITE　〈ステートメント〉

## プットスプライト

**機能**　スプライトパターンを表示します。

**書式**　PUT SPRITE{スプライト面番号}，|(X, Y) / STEP(X, Y)|〔，{カラーコード}〕〔，{スプライトパターン番号}〕

**文例**　PUT SPRITE 1，(100, 100)，7，1

**範囲**　{スプライト面番号}は0～31、座標はX座標が－32～255、Y座標が－32～191。{カラーコード}は0～15、スプライトパターン番号はパターンサイズが8×8のときは0～255、パターンサイズが16×16のときは0～63。

**解説**　あらかじめ定義したスプライトを、指定したスプライト画面に指定した色で表示します。スプライトパターンのサイズはSCREEN文で指定します。

スプライト面は0～31までもっていますが、5つ以上水平方向に並ぶと4つまで表示され、5つめ以降は消されます。

スプライト面の優先順位は{スプライト面番号}の小さい方が高く、大きい方が低くなっています。スプライトが重なるときには{スプライト面番号}の小さい方が表示されます。(手前に重なってみえます。)

スプライトを表示するための座標指定（X, Y）はスプライトパターンの左上角になります。

X座標は－32～255の範囲で指定し、Y座標は－32～191の範囲で指定します。Y座標に208を指定すると、そのスプライト面以降のスプライトパターンはすべて消えますがY座標に209を指定すると、そのスプライト面のスプライトパターンが消えます。

{カラーコード}はスプライトパターンの色になりますが、これを省略すると、COL

OR文で指定した前景色になります。

スプライト番号の指定は、あらかじめSPRITE＄で定義したスプライトパターンの番号になります。

STEP(X, Y)を付けた場合、最終参照点(LP)からの相対指定となります。

**参照**　SPRITE＄, SCREEN

**サンプルプログラム：**

```
10 '130 ハ゛ン カラノ DATA ニ 0 ト 1 テ゛ スキナ エ ヲ カク。
20 '
30 SCREEN 2,3
40 B1$="":B2$="":RESTORE
50 FOR J=1 TO 16
60  READ A$:A=VAL("&B"+A$):IF A<0 THEN A=A+65536!
70  A1=INT(A/256):A2=A-A1*256
80  B1$=B1$+CHR$(A1)
90  B2$=B2$+CHR$(A2)
100 NEXT J:SPRITE$(0)=B1$+B2$
110 PUT SPRITE 0,(100,100)
120 GOTO 120
130 DATA 1111111111111111
140 DATA 1000000000000001
150 DATA 1000000000000001
160 DATA 1000000000000001
170 DATA 1000000000000001
180 DATA 1000000000000001
190 DATA 1000000000000001
200 DATA 1000000000000001
210 DATA 1000000000000001
220 DATA 1000000000000001
230 DATA 1000000000000001
240 DATA 1000000000000001
250 DATA 1000000000000001
260 DATA 1000000000000001
270 DATA 1000000000000001
280 DATA 1111111111111111
```

スプライトを表示するプログラムです。130行～280行の0と1を変更することにより自由な形のスプライトを表示できます。



# BEEP 〈ステートメント〉

ビープ

**機能**　スピーカを鳴らします。

**書式**　BEEP

**文例**　BEEP

**解説**　スピーカを約0.04秒ならします。

CHR＄(7)を PRINT 命令で出力したものと、BEEP 文を実行したものとはまったく同じです。

**サンプルプログラム：**

```
10 FOR J=0 TO 100
20   K=INT(RND(1)*100)
30   FOR L=0 TO K:NEXT L
40   BEEP
50 NEXT J
```

BEEP音を鳴らします。音の間隔は、乱数で決めています。

# PLAY　<ステートメント>
## プレイ

**機能**　音楽を演奏します。

**書式**　PLAY {文字式1}〔,{文字式2}〕〔,{文字式3}〕

**文例**　PLAY "CDEFEDC"

**範囲**　解説を参照

**解説**　ミュージックマクロ命令で表わされた音楽を演奏します。
{文字式1}、{文字式2}、{文字式3}は、それぞれボイスチャンネル1、2、3に対応し、最大3重和音まで出すことが可能です。
文字式がヌルストリング(空)であれば、そのボイスチャンネルは音を出しません。
{文字式}には、以下で述べるミュージックマクロ命令をひとつ、または複数個並べます。文字式は文字定数、文字変数、あるいはそれらを組み合わせた式でもかまいません。
ミュージックマクロ命令は、大きく分けて、音の高さ、音の長さ、音の速さ（テンポ)、音の大きさ、音色、ローカルマクロの6種類があります。

| | | | |
|---|---|---|---|
| A～G | 音の高さ | T | テンポ |
| O | オクターブ | V | 音の大きさ |
| N | 音の高さ | S | エンベロープ形状 |
| L | 音の長さ | M | エンベロープ周期 |
| R | 休　符 | | |

### ●音の高さ

A～G〔#｜+｜-〕　音の高さを音階名(ドレミファソラシ)で表現します。ABCDEFGはハ長調のラシドレミファソに対応します。またA～Gのうしろに#や+記号を付けると半音上がり、-記号を付けると半音下がります。

O{数値}…………上記で説明した音階の音域を1～8の数値で指定します。
初期値はO4で、1点ハ音からの1オクターブを指し、O5はO4の1オクターブ上の音階を、O3はO4の1オクターブ下の音階を指します。
(表1参照)
なおオクターブ指定を行なうと、そのボイスチャンネルで次のオクターブ指定があるまで有効です。

N{数値}…………音の高さを0～96の数値で表現します。N1はO1のC#に相当し、N2、N3……と半音ずつ上がっていき、N95は、O8のBに相当します。
また、N0はR(休符)と同じ働きをします。

表1

O1 O2 O3 O4 O5 O6 O7 O8

N1 N36 N47 N95 N96

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | N37 C⁺ C♯ D⁻ | N39 D⁺ D♯ E⁻ | | N42 F⁺ F♯ G⁻ | N44 G⁺ G♯ A⁻ | N46 A⁺ A♯ B⁻ | |
| N36 C | N38 D | N40 E | N41 F | N43 G | N45 A | N47 B | N48 C |

●**音の長さ**

L {数値}　　音の長さを 1/{数値} 音にします。{数値} の範囲は1～64までで、L1を全音とするとL2以降の長さは次の通りです。

L1　全音符　(4拍)
L2　2分音符　(2拍)
L3　1/3音符　(4/3拍)
L4　4分音符　(1拍)
⋮　⋮
L8　8分音符　(1/2拍)
⋮　⋮
L64　64分音符　(1/16拍)

初期値はL4で、1度音の長さの指定を行なうと、そのボイスチャンネルで次に音の長さの指定があるまで有効です。また1音についてのみ音の長さを変えたいときは、音階名のうしろに音の長さを表わす数値をつけ加えます。たとえばL8EはE8と同じです。

・(ピリオド)……付点音符のように音階名や休符のうしろに付け加えて、本来の音の長

さの1.5倍にします。2個並べて書くと9/4倍、3個並べると27/8倍となります。

R〔(数値)〕……… 休符を表わします。{数値}はLマクロ命令での指定方法と同じで、{数値}を省略するとR4とみなされます。なお、このマクロ命令はLマクロで指定した長さには影響されません。

**●音の速さ(テンポ)**

T{数値}…………テンポを指定します。{数値}は1分間に演奏する4分音符の数を指定し、範囲は32〜255、初期値はT120です。
なお、一度速さの指定を行なうと、そのボイスチャンネルで次の速さ指定があるまで有効です。

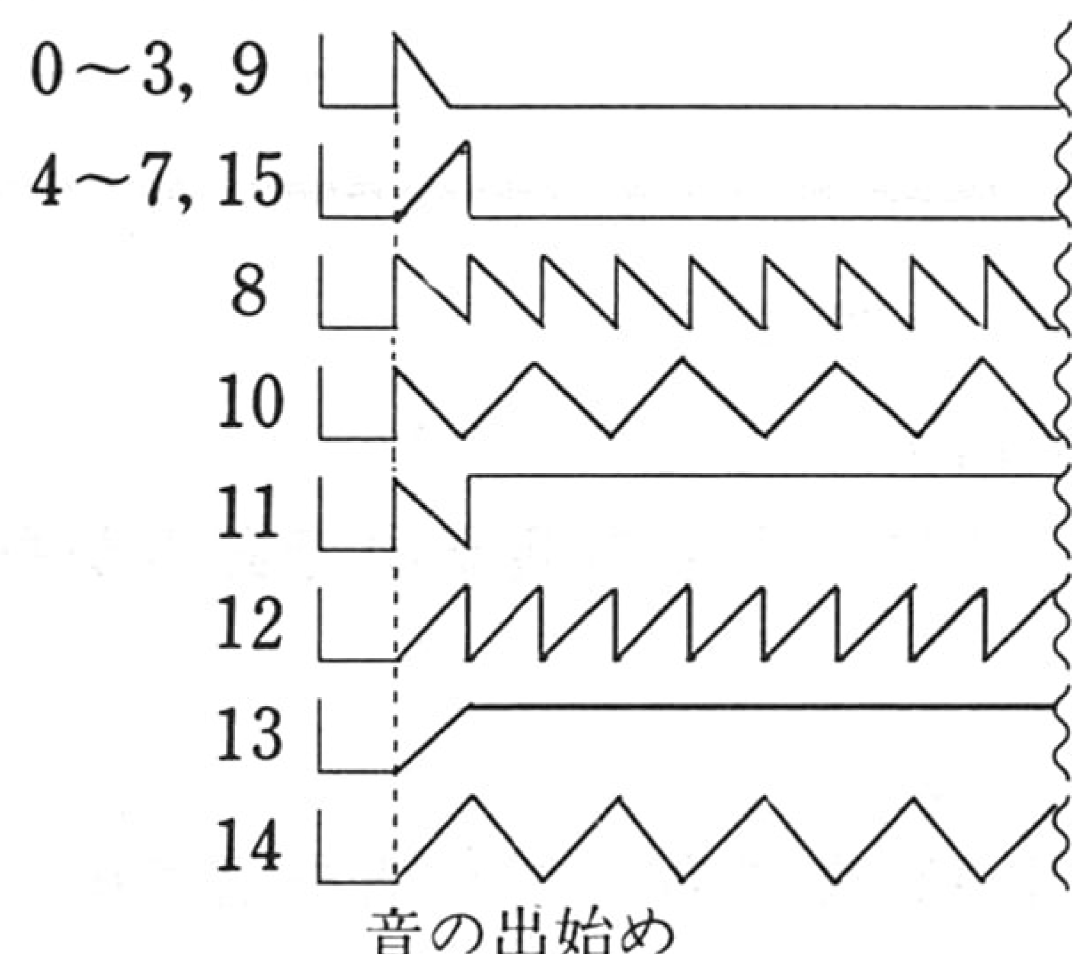
**●音の大きさ(ボリューム)**

V{数値}…………音量の指定をし、{数値}の範囲は0〜15で、数が大きくなるほど音も大きくなります。なお、一度音量の指定を行なうと、そのボイスチャンネルで次の音量指定があるまで有効です。

**●音色**

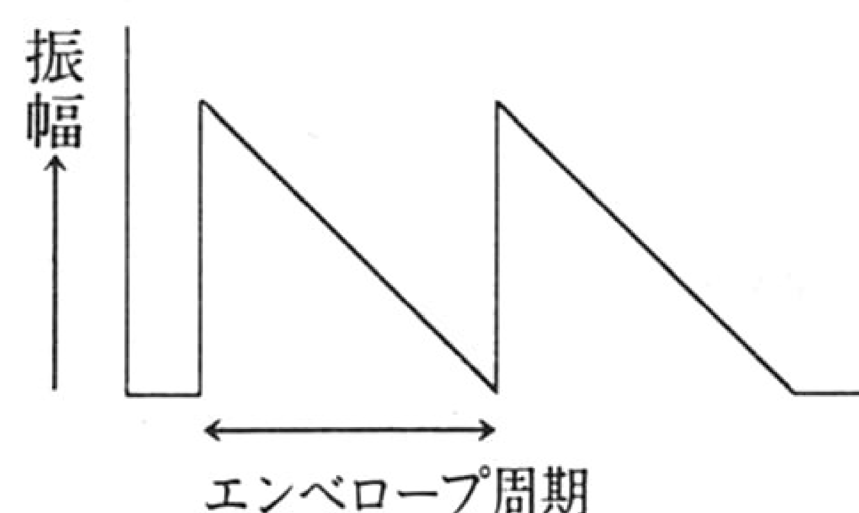
音色はエンベロープ周期(Mマクロ)とエンベロープ形状(Sマクロ)とで、ほぼ決まります。

S{数値}………… エンベロープ形状(音量変化の波形)を指定し、{数値}と波形の対応は次の通りです。




なお、SマクロとVマクロを同時に指定することはできません。

M{数値}…………エンベロープ周期を0〜65535までの{数値}で指定しますが、あまり大きすぎるとエンベロープ形状の特徴が出ません。




エンベロープ形状とエンベロープ周期の組み合わせでいろいろな音を出すことができます。

**●ローカルマクロ**

X{文字変数};……{文字変数}のなかの文字列に含まれているミュージックマクロ命令を実行します。{文字変数}には上記のミュージックマクロをあらかじめ

音を出す

入れておきます。{文字変数}のあとのセミコロン(;)は必ず入れてください。
ローカルマクロは、ミュージックマクロの文字列が255を越える場合や音を少しずつ変化させながら繰り返し演奏するときなどに利用します。
(例)
P$=〝CDECDE〟:Q$=〝GEDCDED〟
PLAY 〝XP$;XQ$;〟

ミュージックマクロ命令では{数値}などのパラメータを定数で指定しますが、〝={変数名};〟の形式にすれば数値変数を指定することができます。
この場合{変数名}のうしろにはセミコロン(;)を必ず入れます。

**サンプルプログラム:**

```
10 PLAY"O4CDEFGABO5C"
20 GOTO 10
```

ドレミファソラシドを演奏します。
CTRL ＋ STOP で中断します。

# PLAY 〈関数〉

プレイ

**機能** 音楽の演奏中であるかどうか調べます。

**書式** PLAY ({ボイスチャンネル})

**文例** A = PLAY(0) その結果 Aには音楽の演奏中であるかを判断し、演奏中ならー1、そうでなければ0が入る。

**範囲** {ボイスチャンネル}は0～3

**解説** 各ボイスチャンネルがPLAY文で命じた音楽を演奏中かどうか判断します。

0:ボイスチャンネル1、2、3、のいずれかが
1:ボイスチャンネル1が
2:ボイスチャンネル2が
3:ボイスチャンネル3が
} 演奏中なら －1 / そうでなければ 0

**サンプルプログラム:**

```
10 DEF FNC$(A)=MID$(STR$(A),2)
20 A$="CDEFGAB"
30 FOR L=2 TO 6
40 IF PLAY(0) THEN 40
50 PLAY"L"+FNC$(2^L):PRINT"1/";2^L,
60 FOR O=1 TO 8
70  PLAY"O"+FNC$(O)+A$
80 NEXT O
90 NEXT L
100 END
```

PRINT文で音の長さを表示すると同時に音を出します。音と表示がずれないように40行で音を出し終ったあとにPLAY文を実行するようにしています。

# SOUND　〈ステートメント〉

サウンド

**機能**　PSG(プログラマブルサウンドジェネレータ)のレジスタに、データを書き込みます。

**書式**　SOUND {レジスタ番号}，{整数式}

**文例**　SOUND　7，37

**範囲**　解説を参照

**解説**　PSGのレジスタを直接書き換えて音を出します。

PSGの音の発生装置と音量の制御装置は3つのチャンネルにわかれているので別々に指定することができます。ただし、ノイズの発生装置とエンベロープの発生装置は、各チャンネルで共用します。

{レジスタ番号} で指定できるレジスタは次の14個です。

| レジスタ | ＼ビット | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|---|
| R 0 | チャンネルA周波数 | 8BIT FT A | | | | | | | |
| R 1 | | | | | | 4BIT CT A | | | |
| R 2 | チャンネルB周波数 | 8BIT FT B | | | | | | | |
| R 3 | | | | | | 4BIT CT B | | | |
| R 4 | チャンネルC周波数 | 8BIT FT C | | | | | | | |
| R 5 | | | | | | 4BIT CT C | | | |
| R 6 | ノイズ周波数 | | | | 5BIT NP | | | | |
| R 7 | チャンネル設定 | | | $\overline{\text{NOISE}}$ | | | $\overline{\text{TONE}}$ | | |
| | | 1 | 0 | C | B | A | C | B | A |
| R 8 | チャンネルA音量 | | | | M | L3 | L2 | L1 | L0 |
| R 9 | チャンネルB音量 | | | | M | L3 | L2 | L1 | L0 |
| R10 | チャンネルC音量 | | | | M | L3 | L2 | L1 | L0 |
| R11 | エンベロープ周期 | 8BIT FT | | | | | | | |
| R12 | | 8BIT CT | | | | | | | |
| R13 | エンベロープ形状 | | | | | E3 | E2 | E1 | E0 |

R 0，R 1————チャンネルAの周波数を2つのレジスタを使って指定する。

R 2，R 3————チャンネルBの周波数を2つのレジスタを使って指定する。

R 4，R 5————チャンネルCの周波数を2つのレジスタを使って指定する。

R 6————ノイズ周波数を指定する。

R 7————各チャンネルから音を出すかどうかを指定する。

R 8，R 9，R10—チャンネルA，B，Cのそれぞれの音量を指定する。

Mを1にするとエンベロープモード、0にするとエンベロープモード解除となる。

L 0～L 3を0にすると無音に、15にすると最大の音量になる。

R11，R12————エンベロープの周期を2つのレジスタを使って指定する。

R13————エンベロープ形状を指定する。

**●周波数の設定**

$$TP = \frac{fclock}{16 \times fT} \qquad CT + \frac{FT}{256} = \frac{TP}{256}$$

fclock　：1.78977MHz
fT　　：出力周波数
FT　　：FTA（R 0）の値
CT　　：CTA（R 1）の値

（例）

440Hz を出力する。

$$TP = \frac{1.78977 \times 10^6}{16 \times 440} \fallingdotseq 254$$

$$CT + \frac{FT}{256} = \frac{254}{256}$$

R 0には　FT = 254
R 1には　CT = 0
を設定します。

● ノイズ周波数の設定

$$NP = \frac{fclock}{16 \times fN}$$

fclock　：1.78977MHz
fN　　：ノイズ周波数(Hz)
NP　　：R 6のNPに設定する値

（例）

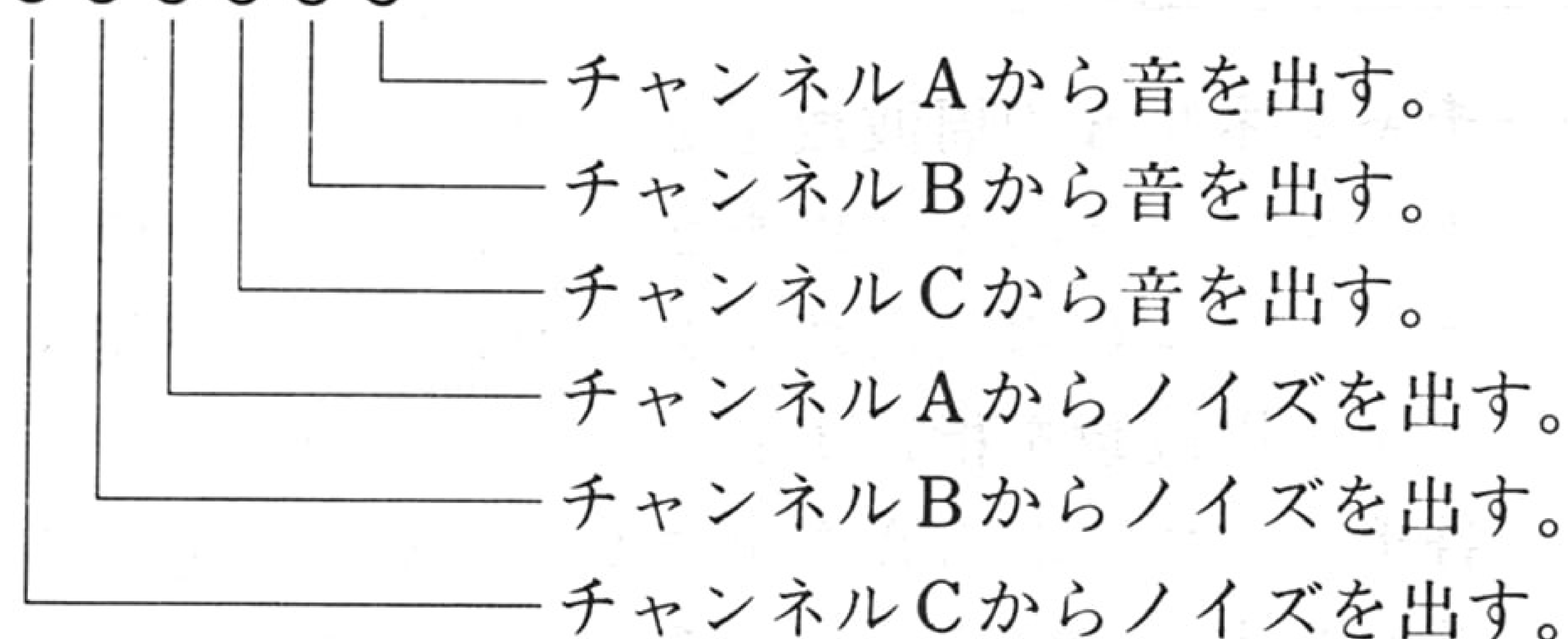
30KHz のノイズ周波数を出力する。

$$NP = \frac{1.78977 \times 10^6}{16 \times (30 \times 10^3)} \fallingdotseq 4$$

R 6には　NP = 4
を設定します。

● チャンネルの設定

使用したいチャンネルに対応するビットを 0 (ゼロ)にする。




（例）

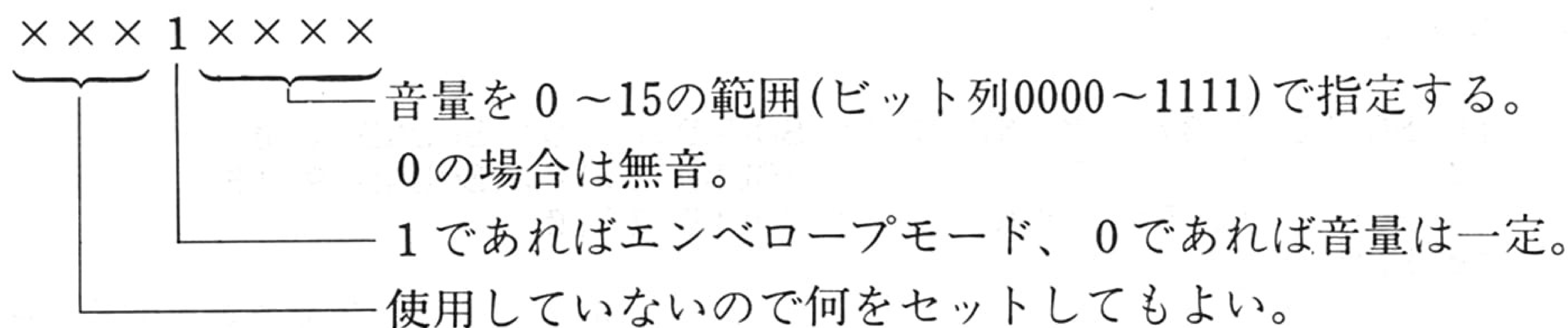
チャンネルA、Cから音を出す(他の動作はしない)。

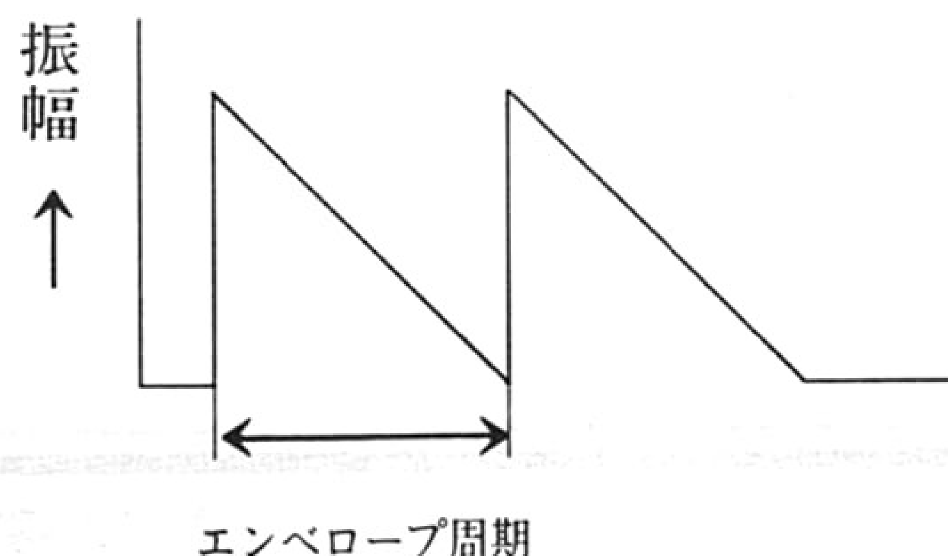
R 7 = &B111010

● 音量設定

音量設定には、大きさが一定のものと、時間的に変化するエンベロープモードのふたつがあります。さらに、エンベロープモードを指定している場合はエンベロープ形状とその周期を指定します。

音を出す




●**エンベロープ周期の設定**

$$EP = \frac{fclock}{256fE} \qquad CT + \frac{FT}{256} = \frac{EP}{256}$$

fclock ：1.78977MHz
fE ：音が増加、減衰する周波数Hz
FT ：FT（R11）に設定する値
CT ：CT（R12）に設定する値

**（例）**

音が減衰して０になるまで２秒とする。

$$EP = \frac{1.78977 \times 10^6}{256 \times 0.5} \fallingdotseq 14000$$

$$CT + \frac{FT}{256} = \frac{14000}{256} = 54 + \frac{176}{256}$$

R11にはFT＝176（10進）を設定する。
R12にはCT＝54（10進）を設定する。

●**エンベロープ形状の設定**

エンベロープ形状は、レジスタR13の下位４ビット(E0～E3)へ０～15（ビット列0000～1111）で指定します。

**エンベロープ発生器出力**

| レジスタの値 | | | | エンベロープ・パターン |
|---|---|---|---|---|
| E3 | E2 | E1 | E0 | E３～E０ |
| 0 | 0 | × | × | |
| 0 | 1 | × | × | |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 1 | |
| 1 | 0 | 1 | 0 | |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 0 | |
| 1 | 1 | 0 | 1 | |
| 1 | 1 | 1 | 0 | |
| 1 | 1 | 1 | 1 | |

→ EP ←エンベロープ周期

**サンプルプログラム：**

```
10 SOUND 7,37: SOUND 6,0: SOUND 0,25: SOUND 1,0
20 SOUND 2,25: SOUND 3,0: SOUND 8,16: SOUND 9,16
30 SOUND 12,50:SOUND 11,7:SOUND 13,0
40 END
```

SOUND文を使って効果音を出すプログラムです。

## TIME 〈システム変数〉

タイム

**機能** 1/60秒ごとに値を1ずつ増やします。

**書式** TIME

TIME＝0

**文例** PRINT TIME

**範囲** TIMEの持つ値は0～65535になります。

**解説** TIMEは1/60秒ごとに1ずつカウントアップされます。また、この範囲の値をTIMEに代入することもできます。

なお、このタイマはVDPが1/60秒ごとに割り込みが発生するのを基準にしていますので、カセットテープ操作中などの割り込み禁止状態では、タイマの値は増えません。

**参照** ON INTERVAL GOSUB

**サンプルプログラム：**

```
10 ' TIME --> TI$ ( XX:XX:XX )
20 CLS
30 DEFINT A-Z
40 DEF FNC$(X)=RIGHT$("00"+MID$(STR$(X),2),2)
50 T!=TIME
60 T1=T!/3600:TT=T!-T1*3600
70 T2=TT/60:T3=TT MOD 60
80 TI$=FNC$(T1)+":"+FNC$(T2)+":"+FNC$(T3)
90 LOCATE 0,10
100 PRINT "TIME:";T!
110 PRINT "TIME$:";TI$
120 GOTO 50
```

時計を表示するプログラムです。

# OPEN
## オープン

〈ステートメント〉

**機能** ファイルを開き、使用可能な状態にします。

**書式** OPEN "{デバイス名}{ファイル名}"〔FOR{モード}〕AS〔#〕{ファイル番号}

**文例** OPEN "CAS:SAMPLE" FOR INPUT AS#1

**範囲** ファイル番号は1から15（MAXFILES 文で指定した値を越えて指定することはできません。）

**解説** {ファイル名}で指定するファイルを開き、INPUT#文やPRINT#文でデータの入出力ができるようにします。オープンするファイルには{ファイル番号}を割り当て、以後のファイルへの入出力では、{ファイル番号}を指定します。

{ファイル名}の形式は"{デバイス名} {ファイル名}"です。

{デバイス名}には次の種類があります。

| デバイス名 | 入・出力デバイス | 使用できるモード | |
|---|---|---|---|
| | | INPUT | OUTPUT |
| CAS: | カセットレコーダー | ○ | ○ |
| CRT: | テキスト画面 | × | ○ |
| GRP: | グラフィック画面 | × | ○ |
| LPT: | プリンタ | × | ○ |

{ファイル名}は1～6文字の文字列で表わしますが、デバイス名がCRT:、GRP:、LPT:の場合には省略してもかまいません。

{モード}はファイルへのアクセス方法を指定します。モードには次の2種類があります。

INPUT　既存のファイルから入力を行ないます。

OUTPUT　新しくファイルを作り、出力を行ないます。

すでにOPENされたファイル番号でOPENしようとすると"File already open"エラーになります。

OPEN文は、指定したファイルを入出力する際に使うバッファ領域を確保し、ファイルをCLOSE文で閉じるときに解放されます。{ファイル番号}は、このバッファ領域と対応しています。

**参照** CLOSE, INPUT#, PRINT#, MAXFILES, INPUT$, EOF

**サンプルプログラム：**

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
30 PSET(100,100),5
40 PRINT #1,INT(RND(1))
50 GOTO 30
```

画面ファイルに文字を表示するプログラムです。同じ位置に表示するため、文字は重なります。

# CLOSE 〈ステートメント〉
クローズ

**機能** ファイルを閉じ、使用不可能状態にします。

**書式** CLOSE〔[#]{ファイル番号}〔,[#]{ファイル番号}〕……〕

**文例** CLOSE #2

**範囲** ファイル番号は1～15

**解説** 使用中のファイルをクローズ(閉じる)します。クローズされたファイルには、読み込みも書き込みもできません。
ファイル番号の指定を省略すると、オープンされているすべてのファイルがクローズされます。
CLOSE文は、ファイルが出力用になっていたときは、バッファに残っている内容をはき出すため、必ずCLOSEを行なわなければなりません。
CLEAR, END, NEW 文を実行すると自動的にCLOSE処理が行なわれます。

# EOF 〈関数〉
イーオーエフ

**機能** ファイルが終了したかどうかを調べます。

**書式** EOF({ファイル番号})

**文例** EOF (1) その結果 この関数の値が－1ならファイルが終りに達している。

**範囲** {ファイル番号}は1～15

**解説** {ファイル番号}で指定したファイルが読み終わったかどうかを調べる関数です。終わりであれば－1(真)、終わりでなければ0(偽)を返します。
この関数はファイルをINPUTモードで開かれたものに関してのみ有効ですので、ファイルが開かれないままEOFをチェックしようとすると“File not OPEN”エラーになります。また、{ファイル番号}が適切でないときは“Bad file number”になります。

**参照** OPEN, INPUT #

# INPUT ＃ 〈ステートメント〉

## インプット クロスハッチ

**機能** ファイルからデータを読み込みます。

**書式** INPUT ＃{ファイル番号}，{変数}〔,{変数}……〕

**文例** INPUT ＃1，Z, X, Y

**範囲** {ファイル番号}は1～15でOPEN文で指定した番号。

**解説** OPEN文でINPUTモードを開いたファイルからデータを読み込み、変数にセットします。INPUT ＃文はデータを読み込む対象がファイルであることを除くとINPUT文とほぼ同じです。

INPUT ＃文で読み込むデータはPRINT ＃文で書き出した各データと対応しています。

{変数}の型はファイルから入力するデータの型と対応している必要があります。

**参照** OPEN, PRINT ＃

**サンプルプログラム**：

```
10 CLS:BEEP
20 PRINT "ﾚｺｰﾀﾞｰ ﾉ ﾛｸｵﾝﾎﾞﾀﾝ ｦ ｵｽ"
30 BEEP:PRINT
40 PRINT "ﾘﾀｰﾝ ｷｰ ｦ ｵｽ"
50 A$=INPUT$(1):IF A$<>CHR$(13) THEN 50
60 OPEN "CAS:DUMMY" FOR OUTPUT AS #1
70 FOR J=1 TO 10
80   PRINT #1,RND(1)
90 NEXTJ
100 CLOSE
110 CLS:BEEP
120 PRINT "ﾚｺｰﾀﾞｰ ﾉ ｽﾄｯﾌﾟﾎﾞﾀﾝ ｦ ｵｽ":BEEP
130 PRINT: PRINT "ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｽ"
140 PRINT "ﾚｺｰﾀﾞｰ ﾉ ｻｲｾｲﾎﾞﾀﾝ ｦ ｵｽ"
150 OPEN"CAS:DUMMY" FOR INPUT AS #1
160 CLS
170 IF EOF(1) THEN CLOSE:GOTO 190
180  INPUT #1,A:PRINT A:GOTO 170
190 PRINT
200 PRINT "ﾚｺｰﾀﾞｰ ﾉ ｽﾄｯﾌﾟﾎﾞﾀﾝ ｦ ｵｽ":BEEP
210 END
```

このプログラムを実行し、表示されるメッセージに従いテープレコーダーの操作をすることによりDUMMYというファイルを作成し次にファイルからデータを読み込みデータを表示します。

# LINE INPUT＃　〈ステートメント〉

ラインインプットクロスハッチ

**機能**　1行単位のデータを、ファイルから文字変数へ読み込みます。

**書式**　LINE INPUT＃{ファイル番号}，{文字変数}

**文例**　LINE INPUT ＃1, Z$, P$

**範囲**　読み込まれるデータは254文字以内。

**解説**　OPEN文でINPUTモードに開いたファイルから1行単位(254文字以内)にデータを読み込むことができます。データが254文字以上のときは、文字変数には254文字まで代入されます。
プログラムをアスキーセーブしたり，シーケンシャルファイルでファイルを書き出したときの区切りはキャリッジリターン (CHR$(13)) とラインフィードコード(CHR$(10))になります。LINE INPUT＃文はCHR$(13)＋CHR$(10)、またはCHR$(13)を区切りとし、その前までをデータとして{文字変数}の値とします。

**参照**　OPEN, CLOSE, PRINT＃

**サンプルプログラム：**

```
10 CLS:BEEP
20 PRINT "ﾚｺｰﾀﾞｰ ﾉ ﾛｸｵﾝﾎﾞﾀﾝ ｦ ｵｽ"
30 BEEP:PRINT
40 PRINT "ﾘﾀｰﾝ ｷｰ ｦ ｵｽ"
50 A$=INPUT$(1):IF A$<>CHR$(13) THEN 50
60 OPEN "CAS:DUMMY" FOR OUTPUT AS #1
70 READ B$:IF B$="END" THEN 100
80   PRINT #1,B$
90 GOTO 70
100 CLOSE
110 CLS:BEEP
120 PRINT "ﾚｺｰﾀﾞｰ ﾉ ｽﾄｯﾌﾟﾎﾞﾀﾝ ｦ ｵｽ":BEEP
130 PRINT: PRINT "ﾃｰﾌﾟ ｦ ﾏｷﾓﾄﾞｽ"
140 PRINT "ﾚｺｰﾀﾞｰ ﾉ ｻｲｾｲﾎﾞﾀﾝ ｦ ｵｽ"
150 OPEN"CAS:DUMMY" FOR INPUT AS #1
160 CLS
170 IF EOF(1) THEN CLOSE:GOTO 190
180 LINE INPUT #1,B$:PRINT B$:GOTO 170
190 PRINT
200 PRINT "ﾚｺｰﾀﾞｰ ﾉ ｽﾄｯﾌﾟﾎﾞﾀﾝ ｦ ｵｽ":BEEP
210 END
220 DATA "I am a boy."
230 DATA "Are you a boy?"
240 DATA "Yes,I am"
250 DATA "END"
```

このプログラムを実行し、表示されるメッセージに従いテープレコーダーの操作をすることによりDUMMYというファイルを作成し次にファイルからデータを読み込みデータを表示します。

# INPUT＄ 〈関数〉

## インプットダラー

**機能** 指定されたファイルから指定された長さの文字を与えます。

**書式** INPUT＄({文字数}〔,〔＃〕{ファイル番号}〕)

**文例** A＄＝INPUT＄(5, ＃2) その結果 ファイル2から5文字読み込んだ文字列が変数A＄に与えられる。

**範囲** {文字数}は1～255

**解説** {ファイル番号}で指定されたファイルから文字列を文字数分だけ読み出します。
{ファイル番号}が省略された場合は、キーボードからの入力になりますが、入力された文字は画面には表示されません。
INPUT＄は指定された{文字数}の文字が入力されるのを待ち続けますが、すでにバッファに入力済みのデータがある場合にはバッファ中から文字を拾ってきます。INPUT＄は、CTRL → C、CTRL → STOPを除くすべての文字をそのまま読み出します。

**参照** INPUT, LINE INPUT

**サンプルプログラム：**

```
10 REM 3ｹﾀ ﾉ ｶｽﾞｱﾃ
20 XX=RND(-TIME/3)
30 X=INT(RND(1)*1000)
40 IF X<100 THEN 30
50 B$=""
60 FOR J=1 TO 3
70  A$=INPUT$(1)
80  IF A$<"0" OR A$>"9" THEN 70
90  B$=B$+A$:PRINT A$;
100 NEXT J
110 IF X=VAL(B$) THEN PRINT " RIGHT!":END
120 IF X<VAL(B$) THEN PRINT " BIG!!":GOTO 50
130 PRINT " SMALL":GOTO 50
```

3桁の数をあてるゲームです。70行でキーボードから1文字ずつ合計3文字を入力しますが入力した数が0～9以外は受けつけないようになっています。

# PRINT＃ 〈ステートメント〉

## プリントクロスハッチ

**機能** ファイルにデータを書き出します。

**書式** PRINT＃{ファイル番号}, {式}〔 ; | , {式} ; | , ……〕

**文例** PRINT ＃1, X, Y, Z

**範囲** {ファイル番号}は1～15

**解説** OUTPUT モードで OPENしたファイルに対して {式} で示す数値や文字列を書き出します。

ファイルが〝GRP:〟、〝CRT:〟のときはPRINTと同じ働きをします。

ファイルが〝LPT:〟のときはLPRINTと同じ働きをします。

ファイルが〝CAS:〟のときは INPUT＃, LINE INPUT＃, INPUT＄ 関数で読むことのできるファイルをカセットテープ上に作成します。

{式} で書き出す数値、文字列はセミコロン(;)、カンマ(,)で区切り、複数の式をつなげることができます。

{式} が数値のときは、数値が文字列におき換えられ、数値と数値の間は空白となります。

{式} が文字のときセミコロン(;)で区切れば、データは空白で区切られずにファイルに書き出されます。このときファイルがOPENされていないと“File not OPEN”エラーになります。

PRINT＃文の最後にカンマもセミコロンも指定しないときは、データの出力後に区切りコードとしてCRコード (&H0D) とLFコード (&H0A) が出力されます。

PRINT＃文の最後にカンマかセミコロンを指定した場合は、CRコードもLFコードも出力しません。

**参照** OPEN, CLOSE, PRINT, INPUT＃, LINE INPUT＃

**サンプルプログラム:**

```
10 SCREEN 2
20 OPEN "GRP:"FOR OUTPUT AS #1
30 PSET (0,0)
40 RESTORE
50 READ A$
60 IF A$="END" THEN FOR J=1 TO 2000:NEXT J:END
70 PRINT#1,A$:GOTO 50
80 DATA "ｸﾞﾗﾌｨｯｸ ｶﾞﾒﾝ ﾆﾓ ﾓｼﾞ ｶﾞ ｶｹﾏｽ"
90 DATA "    ABCDEFGHIJKLMN"
100 DATA "1234567890¥ZQ[TUSXRYGABCDEF"
110 DATA END
```

PRINT＃文を使ってグラフィック画面に文字を表示します。

ファイルの入出力

# PRINT＃ USING 〈ステートメント〉

## プリントクロスハッチユージング

**機能**　文字列や数値を指定された書式でファイルに書き出します。

**書式**　PRINT＃｛ファイル番号｝, USING｛書式｝；｛式｝〔 ；/， ｛式｝ ；/， ……〕

**文例**　PRINT＃2, USING"＃＃＃＃＃"；A

**範囲**　｛ファイル番号｝は1～15

**解説**　｛式｝で表わされる数値や文字列を書式にもとづいてファイルへ書き出します。

｛書式｝の指定方法はPRINT USINGと全く同じなので、そちらを参照してください。

**参照**　PRINT＃, PRINT USING

**サンプルプログラム：**

```
10 SCREEN 2
20 OPEN "GRP:" FOR OUTPUT AS #1
30 PSET(0,0)
40 RESTORE
50 READ U$,A
60 IF U$="END" THEN FOR J=1 TO 2000:NEXT J : END
70 PRINT #1,USING U$;A:GOTO 50
80 DATA "###,###,###",12537443
90 DATA "+#####",-3245
100 DATA "###.######",32.43454
110 DATA END,0
```

PRINT＃USING文を使ってグラフィック画面に文字を表示します。

# INPUT 〈ステートメント〉
インプット

**機能** 指定された変数へデータをキーボードから入力します。

**書式** INPUT〔{"プロンプト文"}{;}〕{変数}〔,{変数}……〕

**文例** INPUT "NAME";P$

**解説** プログラム中にキーボードからの入力を受け付けます。

INPUT 文を実行すると、疑問符(?)と1桁のスペースが画面に表示され、プログラムはキーボードからの入力待ちの状態となります。以降、キーボードから入力されたデータが変数へ代入されます。

{"プロンプト文"}を指定していれば、疑問符の前にプロンプト文が表示されます。

{変数名}で指定した変数はプロンプト文が表示されたあと、リターンキーを押すまでの間にキーインされたデータが入ります。

{変数名}を、カンマ(,)で区切って複数個指定した場合には、入力するデータもカンマで区切って入力します。

対応する{変数名}とデータの型が一致していないと"? Redo from start"を表示して再び入力待ちとなります。(例1)

入力するデータの個数が足りないときは"??"と表示して入力待ちとなり(例2)、入力するデータの個数が多いと"? Extra ignored"が表示され、余分に入力されたデータが無視されます。(例3)

何もしないでリターンキーを押した場合には、変数の値はINPUT文を実行する前と同じです。このときも〈変数〉が複数の場合は、必要数のカンマ(,)だけは入力しなければなりません。

文字変数に文字列を代入する場合で、カンマや文字列の前後の意味のある空白を入力したいときは、引用符(")で文字列を囲む必要があります。この場合は文字として引用符を入力することはできません。

グラフィックモードでINPUT文を使用すると自動的にテキストモードになります。

**(例1)**

```
10 INPUT X
20 PRINT X
```

プログラムを実行し、?に対しＡＢＣ [↵] と操作
(数値変数に文字を入力しようとしている)

**(例2)**

```
10 INPUT A,B$
20 PRINT A,B$
```

プログラムを実行し、1 [↵] と操作
(??と表示しB$の入力要求をする)

**(例3)**

(例2)のプログラムを実行し、1,ABC,3 [↵] と操作
(1がA、ABCがB$に入力され、3は無視される。)

# LINE INPUT 〈ステートメント〉

ライン インプット

**機能** キーボードから1行分全体のデータを区切ることなく入力し、文字型変数に代入します。

**書式** LINE INPUT〔{"プロンプト文"};〕{文字変数}

**文例** LINE INPUT "NAME?";P$

**範囲** キーボードから入力できる文字数は254字までです。

**解説** キーボードからリターンキーが押されるまで入力待ちになります。INPUT 文とちがうのは、カンマ(,)をデータとして入力できることです。また、疑問符(?)は表示されません。

{"プロンプト文"}を指定したときには、入力待ちの前に{プロンプト文}を表示します。実行は CTRL ─ C、または CTRL ─ STOP によって中断されます。

何も入力せず、リターンキーを押したときはヌル(" ")が代入されます。

グラフィックモードの画面で LINE INPUT 文を実行すると、テキストモードに戻ることはありませんが文字変数に正しいデータは入力されません。

**参照** LINE INPUT #, INPUT

**サンプルプログラム:**

```
10 LINE INPUT A$
20 A=INSTR(A$,",")
30 IF A=0 THEN PRINT A$:GOTO 10
40 PRINT LEFT$(A$,A-1)
50 A$=MID$(A$,A+1)
60 GOTO 20
```

このプログラムを実行し、文字列を入力すると、その文字列の中にあるカンマ(,)の位置で改行してカンマをのぞき入力された文字列を表示します。
CTRL ─ STOP でプログラムを終了させます。

# INKEY$ 〈関数〉

インキーダラー

**機能** キーが押されていればその文字を、押されていなければヌルストリングを与えます。

**書式** INKEY$

**文例** A$ = INKEY$ 押されたキーの文字をA$に代入する。

**解説** キーが押されていない状態(キーボードバッファが空)であれば、INKEY$の値はヌルストリングになります。

キーが押されている状態(キーボードバッファが空いていない)であれば、バッファの先頭から1文字取り出し、関数の値はその文字になります。そのため、INKEY$が実行される以前に押されたキーがキーボードバッファに残されていた場合、INKEY$により読み込まれる場合があります。

キーボードバッファのクリアーは、10 IF INKEY$ 〈 〉 〃 〃 THEN 10 により実行できます。

なお、CTRL ─ C、CTRL ─ STOP はINKEY$関数によって与えることはできません。

押されたキーは画面上に表示されません。

INPUT$(1)とちがい、キー入力待ちにはならないので通常IF文と組み合わせて用います。

**参照** INPUT, LINE INPUT, INPUT$

**サンプルプログラム:**

```
10 A$=INKEY$
20 IF A$<>"" THEN PRINT A$;
30 GOTO 10
```

入力した文字を次々と表示します。
CTRL ─ STOP で終了します。

# LPRINT/LPRINT USING 〈ステートメント〉

エルプリント／エルプリント ユージング

**機能** 文字列や数値を指定された書式でプリンタに出力します。

**書式** LPRINT〔{式}…〕

LPRINT USING {書式}; {式}……

**文例** LPRINT P$, X

LPRINT USING "####"; X

**解説** {式} で表わす文字列や数値をプリンタに出力します。

複数個の {式} を指定する場合は {式} の間に、カンマ( , )、またはセミコロン( ; )を入れて区切ります。

使用法は PRINT/PRINT USING を参照してください。

**参照** PRINT, PRINT USING

**サンプルプログラム：**

```
10 INPUT "A=";A
20 INPUT "B$=";B$
30 LPRINT A,B$
40 LPRINT USING "### & &";A;B$
50 END
```

LPRINT USINGを使い、桁ぞろえをして、プリンタに印字します。

# LPOS 〈システム変数〉

エルポジション

**機能** 現在のプリンタのヘッドの位置を与えます。

**書式** LPOS ( {式} )

**文例** A = LPOS (0)

**範囲** {式} はダミーの引数で、変数、数式でもかまいません。

**解説** プリンタのヘッド位置を与えます。これは、プリンタに打ち出すためのプリンタバッファの値のため、実際のプリンタのヘッド位置とは必ずしも合うとは限りません。

**参照** WIDTH, LPRINT

## MOTOR　〈ステートメント〉
### モーター

**機能**　カセットテープレコーダーのモーターの ON/OFF を制御します。

**書式**　MOTOR [ {ON | OFF} ]

**文例**　MOTOR ON

**解説**　MSX 内蔵のモーター用リレーを ONまたはOFFにします。

ON/OFF を指定しない場合は、ON 状態ならば OFF に、OFF 状態ならばONにします。

MOTOR コマンドはリモート端子付のカセットテープレコーダーでカセットテープを巻きもどすときなどに使用します。但しこの命令を使わなくても、早送りや巻きもどしのできるカセットテープレコーダーもあります。

**サンプルプログラム：**

```
10 IF INKEY$<>CHR$(13) THEN 10
20 MOTOR
30 GOTO 10
```

RETURNキーの入力があるごとにMOTORのON, OFFを切り換えます。
CTRL + STOP で終了します。

# PAD　　〈関数〉

パッド

**機能**　タブレットの状態を調べます。

**書式**　PAD（{式}）

**文例**　A ＝ PAD(3) その結果 Aにはタブレットが押されていないとき0、押されているとき－1の値が入る。

**範囲**　式の値は、0～7の整数値。

整数式は0～3のとき、ジョイスティック端子1（ポート1）に接続したタブレット。

4～7のとき、ジョイスティック端子2（ポート2）に接続したタブレット。

**解説**　{式}の値を0または4にすると、タブレットがペンで押されているかどうかが返されます。

－1：押されている。

0：押されていない。

{式}の値を1または5にすると、タブレットが押されている点のX座標が返されます。タブレットが押されていない場合、0が返されます。

{式}の値を2または6にすると、タブレットが押されている点のY座標が返されます。タブレットが押されていない場合、0が返されます。

{式}の値を3または7にすると、タブレット上のスイッチが押されているかどうかが返されます。

－1：押されている。

0：押されていない。

**サンプルプログラム**：

```
10 SCREEN 2
20 F=PAD(0): IF F=0 THEN FF = 0 : GOTO 20
30 X=PAD(1): Y=PAD(2)
40 IF FF THEN LINE-(X,Y) ELSE PSET(X,Y)
50 FF=-1: GOTO 20
```

タブレット（タッチパネル）をジョイスティック端子1に接続して、タブレットにより画面に線が描けます。CTRL＋STOPでプログラムを終了させます。

# STICK 〈関数〉

## スティック

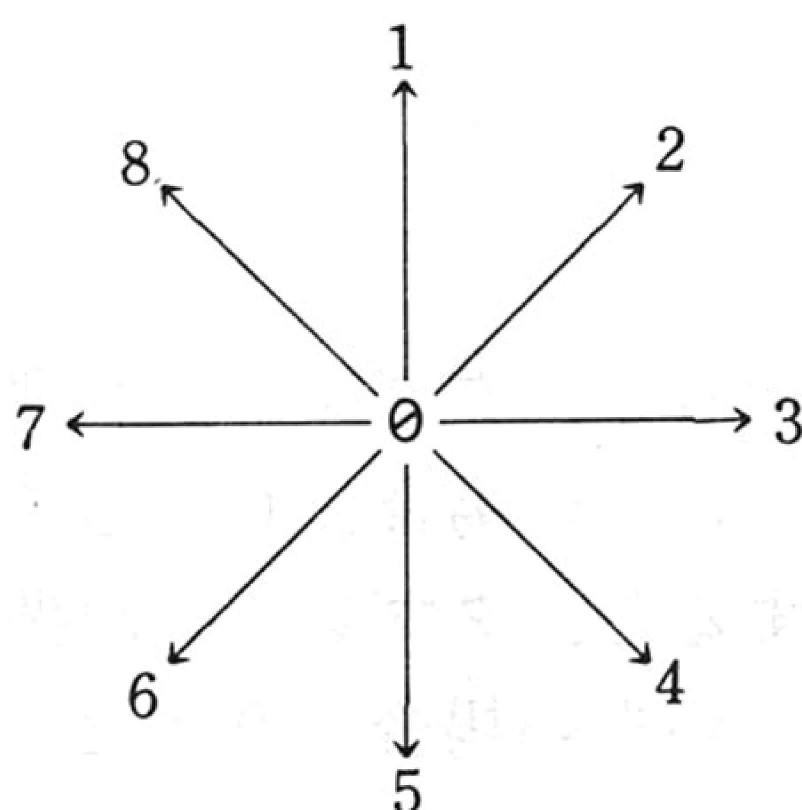
**機能**　ジョイスティックの状態を与えます。

**書式**　STICK(〈ジョイスティック番号〉)

**文例**　A＝STICK(0) [その結果] 変数Aにカーソルキーの状態を示す値が入る。

**範囲**　〈ジョイスティック番号〉は0～2。0はカーソルキー、1はポート1につながったジョイスティック、2はポート2につながったジョイスティック。

**解説**　ジョイスティックがどの方向に押されているかを調べます。どちらの方向にも押されていないときは0が与えられます。




カーソルキーのときは[↑]が1、[→]が3、[↑]と[→]キーが同時に押されたときが2、[↓]が5、[→]と[↓]キーが同時に押されたときが4になります。

**サンプルプログラム：**

```
10 INPUT "STICK No.=";S
20 IF S<0 OR S>2 THEN 10
30 CLS
40 LOCATE 5,5:PRINT "JOY STICK TEST"
50 LOCATE 6,7:PRINT "STICK No.=";S
60 LOCATE 10,9:PRINT STICK(S)
70 GOTO 60
```

ジョイスティック番号を入力し、このスティック番号をもとにスティックの値を表示します。

# STRIG 〈関数〉

## エストリガー

**機能** ジョイスティックのトリガーボタンが押されたかどうかを調べます。

**書式** STRIG(｛ジョイスティック番号｝)

**文例** A=STRIG(1) その結果 ジョイステイック1のトリガーボタン1が押されているとき変数Aに−1が入り、押されていないときは、0が入る。

**解説**

| ジョイステック番号 | ポート番号 | トリガーボタン |
|---|---|---|
| 0 | | スペースキー |
| 1 | 1 | 1 |
| 2 | 2 | 1 |
| 3 | 1 | 2 |
| 4 | 2 | 2 |

｛ジョイスティック番号｝が0のときはスペースキー、1はポート1に接続されたジョイスティックのトリガーボタン1、2はポート2のジョイスティックのトリガーボタン1、3はポート1のトリガーボタン2、4はポート2のトリガーボタン2。

｛ジョイスティック番号｝で指定したジョイスティックのトリガーボタン（本体のトリガーキー）、またはスペースキーの状態を調べます。

トリガーボタンが押されているときは−1、押されていないときは0が返ってきます。

**参照** STRIG ON／OFF／STOP, ON STRIG GOSUB

**サンプルプログラム：**

```
10 A$="CDEFGAB"
20 PRINT "PRESS SPACE BAR"
30 X=INT(RND(1)*7)+1
40 IF STRIG(0) THEN PLAY "L8"+MID$(A$,X,1) ELSE 30
50 IF STRIG(0) THEN 50 ELSE 30
```

スペースキーが押されたかどうかをSTRIG関数で調べています。押されているならば音を出します。CTRL → STOP で中断します。

# PDL 〈関数〉

パドル

**機能** パドルの状態を与えます。

**書式** PDL（{パドル番号}）

**文例** A = PDL(1) [その結果] Aには0～255の数値が入る。

**範囲** パドルがポート1につながっているときは{パドル番号}は、1、3、5、7、9、11、ポート2ならば2、4、6、8、10、12

**解説** パドルの状態は回転の度合によって0～255の範囲の数値で得られます。

**サンプルプログラム：**

```
10 CLS:LOCATE 3,5:PRINT "  *** ﾊﾟﾄﾞﾙ ﾉ ﾃｽﾄ ***"
20 LOCATE 8,7:INPUT "ﾊﾟﾄﾞﾙ No. = ";P:PRINT
30 IF P<1 OR P>12 THEN 20
40 PRINT "    ｵﾜﾙ ﾄｷ ﾊ ﾅﾆｶ ｷｰ ｦ ｵｽ"
50 PRINT
60 LOCATE 12,12
70 PRINT USING"###";PDL(P)
80 IF INKEY$="" THEN 60
90 END
```

パドルのテストを行なうプログラムです。パドルの番号(1～12)を入力したあとにパドルの回転を示す値を表示します。

※ジョイパッドやジョイスティックでは0から255のどちらかの結果しか表示しません。

# INP 〈関数〉

インプ

**機能** 指定された入力ポートの値を与えます。

**書式** INP（{ポート番号}）

**文例** A = INP（&H31）変数Aにポートアドレス31Hで指定されるI/Oポートから読み込んだ値をセットする。

**範囲** ポート番号は0～255

**解説** {ポート番号}で指定された入力ポートから1バイト(8ビット)のデータを読み取り、それを関数値とします。

ポート番号の指定を誤まると"Illegal function call"エラーとなります。

**参照** OUT

**サンプルプログラム：**

```
10 OUT 170,(INP(170)AND&HF0)OR8
20 A=INP(169)
40 IF(A AND 1)=0 THEN PRINT "SPACE";
50 IF(A AND 2)=0 THEN PRINT " HOME";
60 IF(A AND 4)=0 THEN PRINT " INS ";
70 IF(A AND 8)=0 THEN PRINT " DEL ";
80 PRINT:GOTO 10
```

ポートを読んで、どのキーが押されたか判断するプログラムです。

## OUT 〈ステートメント〉
アウト

**機能** 指定されたポートに1バイトのデータを送ります。

**書式** OUT {ポート番号}, {式}

**文例** OUT &H52, &H15

**範囲** {ポート番号}, {式}ともに0～255の整数式

**解説** {ポート番号}は出力ポートの番号、そして{式}は出力する1バイトのデータで、出力ポートに{式}の値を送ります。

(注) OUT文を不用意に実行すると、電源を切る以外に回復できなくなることがありますから注意してください。

## WAIT 〈ステートメント〉
ウェイト

**機能** コンピュータの入力ポートをモニターする間、プログラムの実行を停止します。

**書式** WAIT {ポート番号}, {式1}〔, {式2}〕

**文例** WAIT 4, 8, 225

**範囲** {ポート番号}は、0～255、{式1}、{式2}とも0～255

**解説** この命令を実行すると、指定したポートからの入力データのビットパターンが、{式}で指定した状態になるまで、プログラムを中断します。
{ポート番号}で指定したポートから読み込んだデータと{式2}の値のXORをとり、{式1}とのAND演算をします。その結果が0 (偽) でなくなるまでポートの状態を読み続けます。結果が0でなければ (真)、プログラムの実行は次の行に移ります。{式2}が省略された場合は0とみなされます。

(注) WAIT文の実行により、無限ループにはいってしまう場合がありますので使用には充分気をつけてください。

# PEEK　〈関数〉
ピーク

**機能**　メモリーの指定された番地の内容を与えます。

**書式**　PEEK（{番地}）

**文例**　A = PEEK（&H8000）［その結果］Aには8000H番地の内容が入る。

**範囲**　{番地}は、－32768～65535（&H0～&HFFFF）の値を持つ数式

**解説**　{番地}で指定されたメモリーの内容を読み出します。読み出されるデータは1バイトの値(0～255)です。

PEEK関数はPOKE文と対応している命令です。

**参照**　POKE

**サンプルプログラム：**

```
10 INPUT "ADDRESS=";A$:AD=VAL("&H"+A$)
20 FOR A=AD TO AD+120 STEP 8
30  PRINT:PRINT RIGHT$("000"+HEX$(A),4);"  ";
40  FOR B=A TO A+7
50   PRINT RIGHT$("0"+HEX$(PEEK(B)),2);" ";
60  NEXT
70 NEXT
80 PRINT:GOTO 10
```

メモリーの内容を128バイト単位で表示するプログラムです。
※30行〃　〃内のスペースは2個
　40行〃　〃内のスペースは1個

# POKE　〈ステートメント〉

ポーク

**機能**　メモリー上に指定された番地にデータを書き込みます。

**書式**　POKE {番地}，{データ}

**文例**　POKE &HE000，&HDD

**範囲**　{番地}は、−32768〜65535(&H 0 〜&HFFFF)、{データ}はメモリーに書き込まれるデータで 0 〜255(&H 0 〜&HFF)

**解説**　指定されたメモリー上の番地に1バイト（8ビット）のデータを書き込みます。
この命令は、現在のメモリーの内容を書き換えてしまうため、不用意に使うと、BASIC が使っている作業領域をこわしてしまい、作動できなくなります。
POKE 文の逆の働きをする関数として PEEK 関数があります。

**参照**　PEEK

**サンプルプログラム**：

```
10 CLEAR 100,&HC7FF
20 FOR J%=&HC800 TO &HDFFF
30 PRINT HEX$(J%),
40 FOR K%=0 TO 255
50   POKE J%,K%
60   IF PEEK(J%)<>J% THEN PRINT "ERROR RAM AT ";HEX$(J%):END
70 NEXT K%
80 NEXT J%
90 PRINT "ALL RIGHT (C800-DFFF)"
100 END
```

C800番地からDFFF番地のRAMをテストするプログラムです。0 〜255の値を書き込んだあとに読み出し、エラーがないかどうかをチェックしています。

# USR 〈関数〉

ユーザ

**機能** 機械語で作られたサブルーチンを呼び出します。

**書式** USR〔｛番号｝〕(｛引数｝)

**文例** J＝USR 2(0) [その結果] USR2関数を呼び出して返される値になる。

**範囲** ｛番号｝は0～9、省略した場合は0になります。

**解説** 機械語のプログラムを実行します。｛番号｝の0～9は、DEF USR文で定義した番号と対応します。

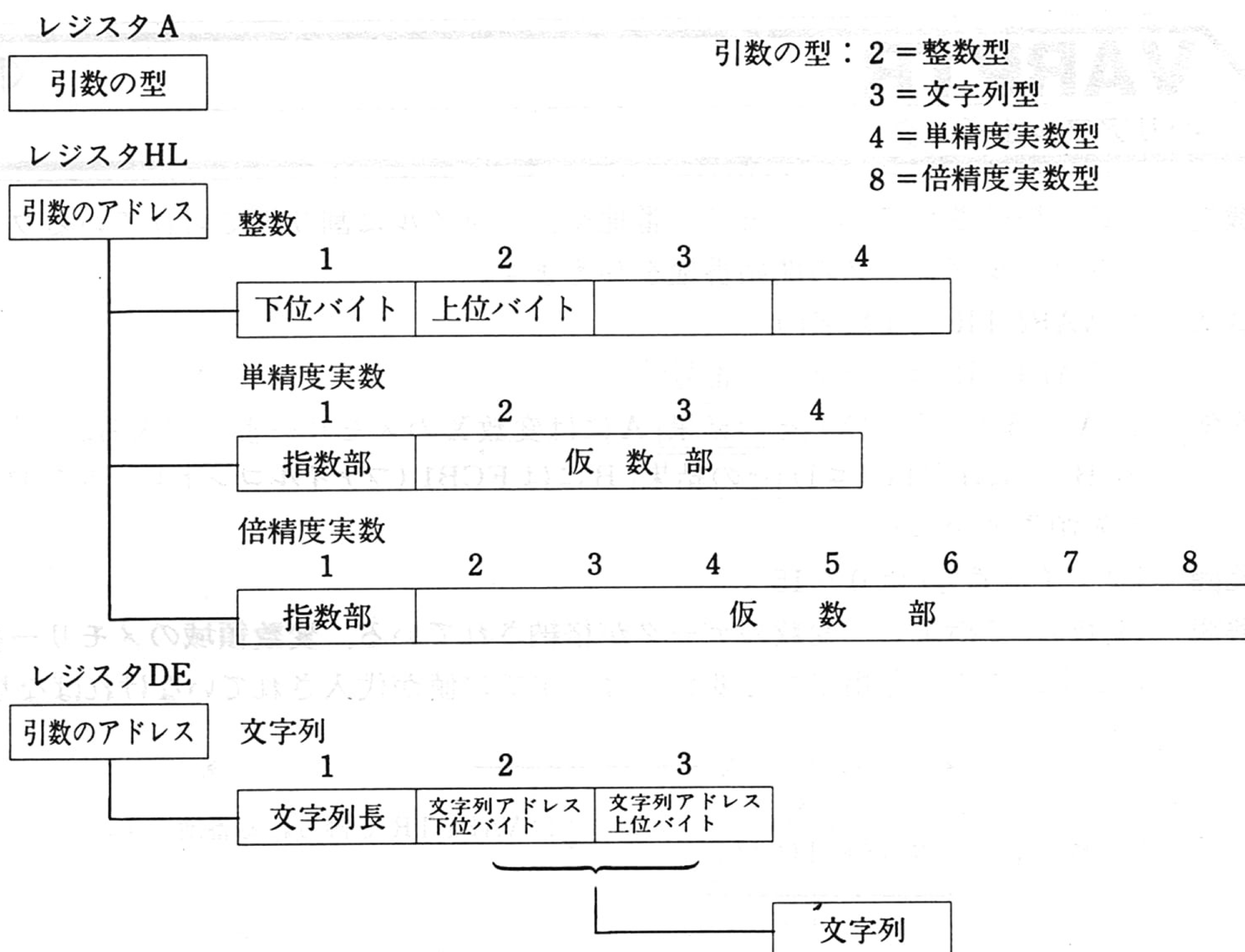
｛引数｝はBASICから機械語ルーチンへの値の受け渡しをします。なお､引数は省略できませんので、呼び出すルーチンに引数を渡す必要がないときには、必ずダミーの引数を指定してください。

USR関数を実行すると、レジスタAに引数の型がセットされ、またHLには引数のアドレス（文字列の場合にはDEストリングディスクリプタのアドレス）がセットされ、機械語ルーチンからの参照ができるようになります。

機械語ルーチンでの処理の結果を渡すときは、引数の内容を書き換えます。このとき、呼び出しに指定した｛引数｝と異なる型に変更することはできません。また、文字列の場合は文字列の長さを変更することはできません。

- 整数型の数値は2バイトの2進形式で表わされ、下位バイト、上位バイトの順で格納されます。
- 単精度実数型の数値は4バイトで表わされ、先頭1バイトが指数部、そのあとに3バイトの仮数部が続きます。指数部の左側1ビットめが0ならば、仮数部が正、1ならば仮数部が負です。以後7ビットがE＋62～E－64までの指数を表わします。また、仮数部は1桁を4ビットで表わす6桁の2進化10進数で表わされています。
- 倍精度実数型の数値は8バイトで表わされ、先頭1バイトが指数部、そのあとに7バイトの仮数部が続きます。

機械語ルーチンからの戻りはRET命令を使います。




**参照** DEF USR

**サンプルプログラム：**

```
10 CLEAR 200,&HDFFF
20 FOR J=&HE000 TO &HE008
30 READ A$:POKE J,VAL("&H"+A$)
40 NEXT J
50 DEF USR=&HE000
60 X%=&H1234
70 D%=USR(X%)
80 PRINT HEX$(D%)
90 END
100 DATA 23,23,56,23,5E,72,2B,73,C9
```

X％の値の上位２桁と下位２桁を入れかえるプログラムです。整数型の引数を使っています。

# VARPTR 〈関数〉

## バリアブルポインタ

**機能** 変数の格納されているメモリー番地や、ファイルに割り当てられているファイルコントロールブロックの開始番地を与えます。

**書式** (1) VARPTR（{変数名}）

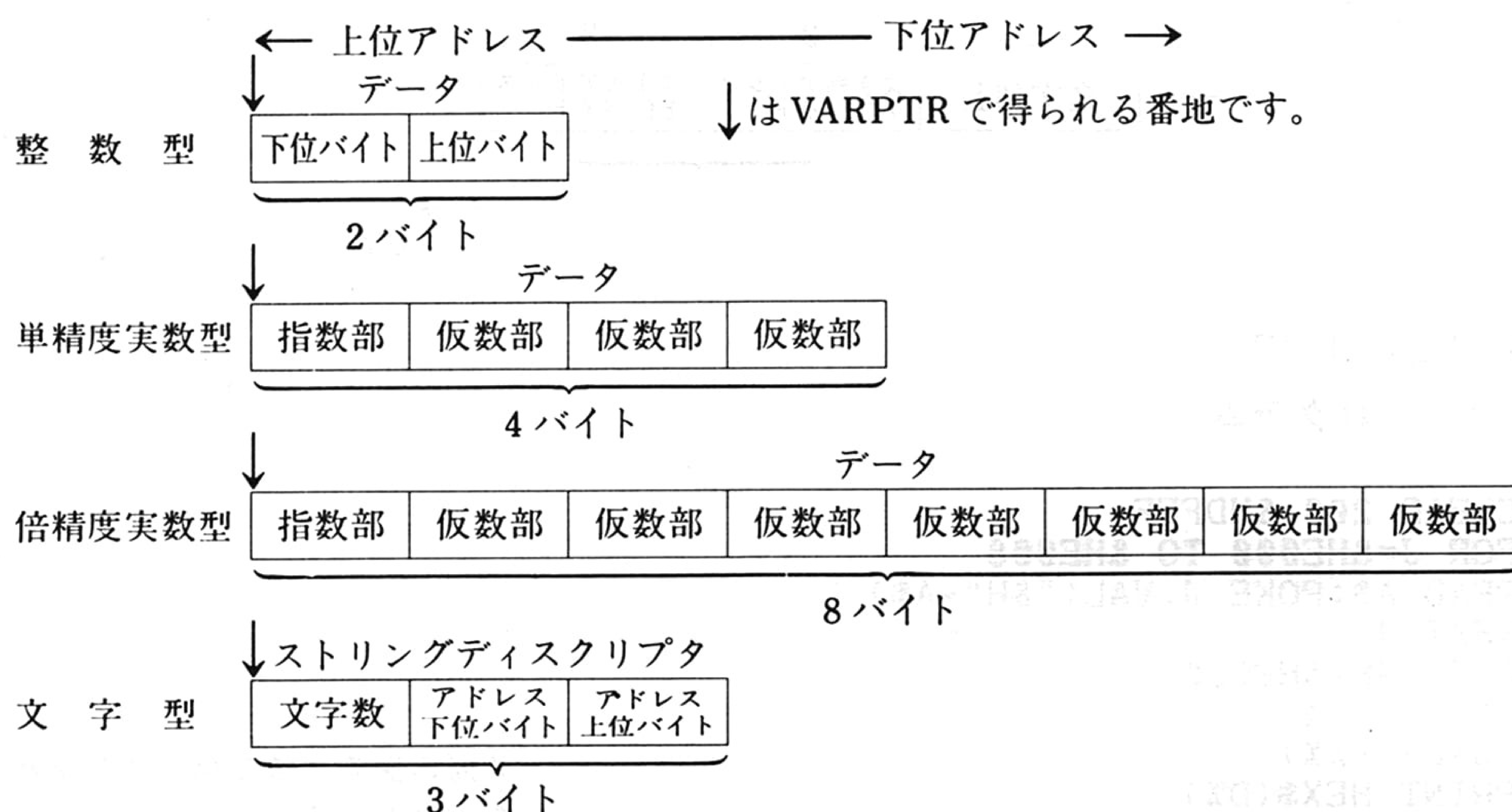
(2) VARPTR（＃{ファイル番号}）

**文例** (1) A = VARPTR(X) [その結果] Aには変数Xのメモリー番地が入る。

(2) B = VARPTR(#1) [その結果] BにはFCB1（ファイルコントロールブロック1）の先頭番地が入る。

**範囲** {ファイル番号}は0～15

**解説** {変数名}で指定した変数のデータが格納されている、変数領域のメモリー番地を求めます。このとき指定する変数には、すでに値が代入されていなければなりません。




指定した{ファイル番号}に割り当てられたFCB（ファイルコントロールブロック）の値を与えます。

**サンプルプログラム：**

```
10 DEFINT A-Z
20 J=0
30 K=VARPTR(J)
40 POKE K,10
50 PRINT J
60 END
```

POKE文でJの変数を書きかえるプログラムです。

# 第3章 音楽の使い方

MSX BASICは､音楽の演奏を３重和音１ノイズで行なうことができます。そのための命令がPLAY文です。では、PLAY文の指定の方法を説明します。

## 3-1 3重和音の出し方

PLAY文で音楽を自動演奏するには、次の文字式を指定します。

PLAY {音声１の文字式}〔, {音声２の文字式}, {音声３の文字式}〕

{音声nの文字式} は､ミュージックマクロ言語を使った文字列で指定します。このとき音声文字式の内容を誤って定義すると“Illegal function call”エラーとなります。

実際に音を出してみましょう。音声１からドレミ、音声２からレミファ、音声３からミファソ、の音を出します。それには次のようにします。

PLAY "cde", "def", "efg"

上の文字式に、c､d､e､f､gが使われていますが、これらはそれぞれ ド､レ､ミ､ファ､ソ の音になります。このように、MSXでは、それぞれの音をアルファベットに対応させています。次の表を参照してください。 ※アルファベットは大文字でも可

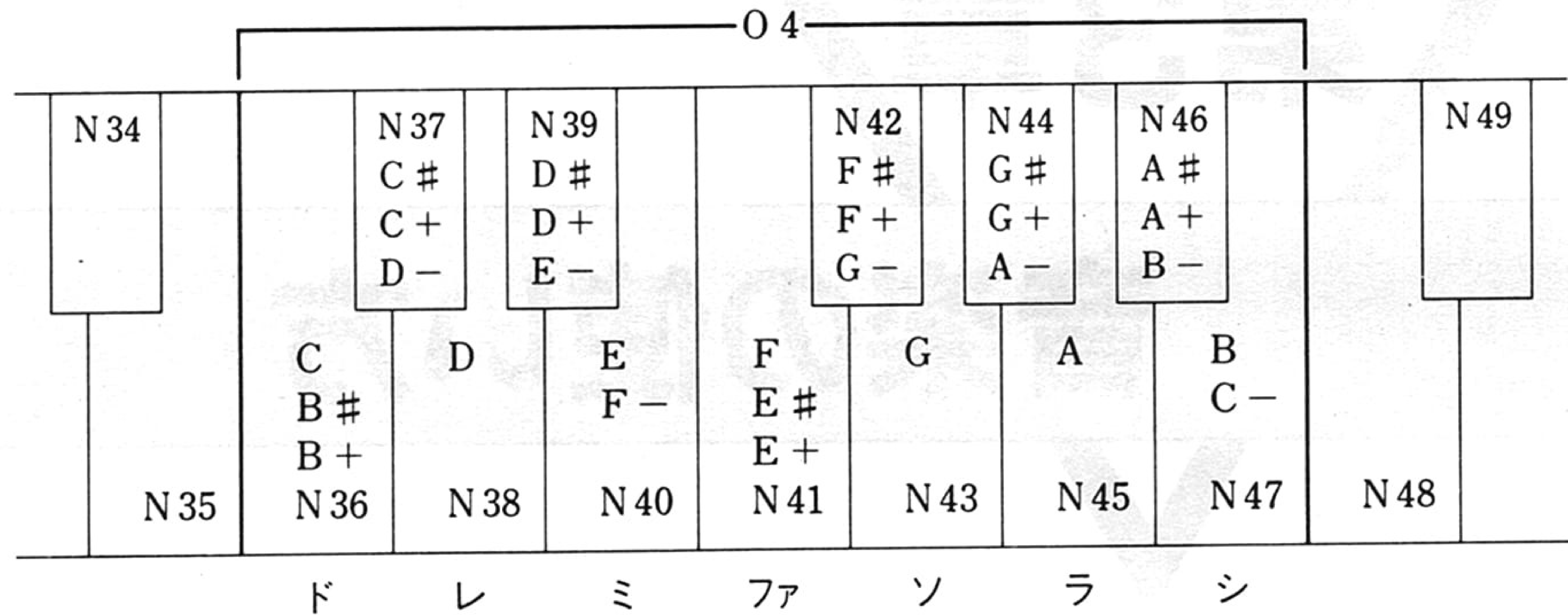
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ハ | ニ | ホ | ヘ | ト | イ | ロ | |
| c | d | e | f | g | a | b | |
| ド | レ | ミ | ファ | ソ | ラ | シ | ハ長調 |

c､d､e､f､g､a､bが、それぞれハ、ニ、ホ、ヘ、ト、イ、ロ、ハ長調のド、レ、ミ、ファ、ソ、ラ、シに対応しています。

また、a～gのあとに＃か＋を付けると半音上がり（シャープ）、－を付けると半音下がります（フラット）。

## 3-2 けん盤と対応してみると

けん盤に対応してみると次のようになります。




オクターブの指定は、〝O{n}〞となります。つまりOのあとに数字を付けることにより、オクターブを自由に設定できます。{n}は1～8で、O4はハ長調のドになります。このとき{n}を省略するとOの値は4になります。では、次のように指定するとどのような音になるでしょう。実際にやってみてください。

PLAY〝O4cdefgab O5cdefgab O6c〞

次に、音の高さを絶対的な数値で指定するには〝N{n}〞を使います。nは1～96まで指定できます。N36はO4Cと同じ高さです。N0は休符になり、あとで述べるRと同じ働きをします。前の図も参考にしてください。

## 3-3 音の長さの指定

音の長さはLで指定します。

L{n}

{n}は1～64で、音の長さは全音符の1/(n)になります。また、特定の音だけ長さを変えるには、音名(A～G)のあとに{n}を指定します。

| | | | |
|---|---|---|---|
| L1 | 全音符（𝅝） | L16 | 16分音符（𝅘𝅥𝅯） |
| L2 | 2分音符（𝅗𝅥） | L32 | 32分音符（𝅘𝅥𝅰） |
| L4 | 4分音符（♩） | L64 | 64分音符（𝅘𝅥𝅱） |
| L8 | 8分音符（♪） | | |

では、休符はどうでしょうか。休符はRで指定します。

R{n}
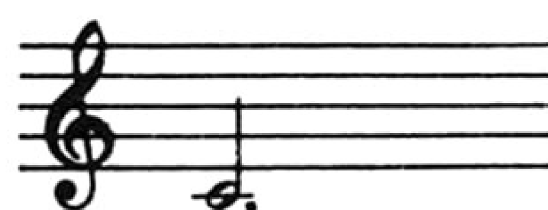
{n}は長さで、1～64の範囲になります。R{n}の長さとL{n}の長さは同等です。{n}を省略するとR4と同じになります。

| | | | |
|---|---|---|---|
| R1 | 全休符（𝄻） | R16 | 16分休符（𝄿） |
| R2 | 2分休符（𝄼） | R32 | 32分休符（𝅀） |
| R4 | 4分休符（𝄽） | R64 | 64分休符（𝅁） |
| R8 | 8分休符（𝄾） | | |

ピリオド(・)は付点を表わし、・ひとつにつき音符の長さを3/2倍します。A～G、N、Rのあとにつけることができます。ただしLのあとにつけることはできません。

C2.　付点2分音符(𝅗𝅥.)

R4.　付点4分休符(𝄽.)



## 3-4 音楽のテンポと大きさ

音楽演奏のテンポは

T{n}

で表わします。これは1分間あたりの4分音符の数を設定するもので、{n}は32～255になります。{n}を省略すると120になります。

また、音の大きさ(ボリューム)は

V{n}

で表します。{n}は1～15で、15がいちばん大きい音になります。省略すると8になります。Vを指定することにより、エンベロープの指定を解除します。

## 3-5 音楽の演奏中をチェックするには

音楽の演奏中に、プログラムで他の動作を行なうのは、何も気にしないでできますが、音楽の演奏が終わってから次の処理をしたいときがあります。そのために、演奏が終わったかどうかをチェックしなければなりません。このようなときPLAY関数を使います。

PLAY関数は、引数が0の場合は音声の全てをチェックし、1、2、3の場合はそれぞれ音声1～3を表わします。結果として、演奏中であれば－1、そうでなければ0になります。

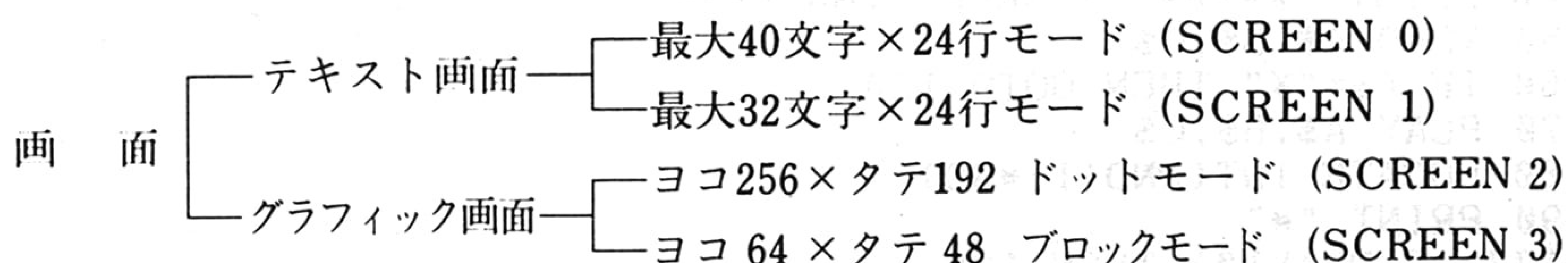
```
10 COLOR 11,1,1
20 R=RND(-TIME)
30 CLS
40 PRINT "***** TWINKLE STAR *****"
50 READ A$,B$,C$
60 IF A$="¥" THEN GOTO 120
70 PLAY A$,B$,C$
80 LOCATE INT(RND(1)*39)
90 PRINT "*"
100 IF PLAY(0) THEN 100
110 GOTO 50
120 PRINT "***** THE END *****"
130 FOR J=1 TO 1000:NEXT J
140 COLOR 15,4,7:CLS
150 END
160 '
170 ' DATA
180 '
190 DATA V12,V10,V10
200 DATA O4L4CCGGAAG.
210 DATA O3L2CCCL4C..
220 DATA O3L8R8GEGR8GEGR8AFAR8GE
230 DATA O4L4FFEEDDC.
240 DATA O2L2BO3CO2GO3L4C..
250 DATA O3L8R8GDGR8GEGR8GFGR8GE
260 DATA O4L4GGFFEED.
270 DATA O3L2CO2BO3CO2L4G..
280 DATA O3L8R8GEGR8GDGR8GEGR8GD
290 DATA O4L4GGFFEED.
300 DATA O3L2CO2BO3CO2L4G..
310 DATA O3L8R8GEGR8GDGR8GEGR8GD
320 DATA O4L4CCGGAAG.
330 DATA O3L2CCCL4C..
340 DATA O3L8R8GEGR8GEGR8AFAR8GE
350 DATA O4L4FFEEDDC.
360 DATA O2L2BO3CO2GO3L4C..
370 DATA O3L8R8GDGR8GEGR8GFGL4C.
380 DATA "¥"
390 DATA "¥"
400 DATA "¥"
```

このプログラムは画面上に〝＊〟を表示しながら「きらきら星」を３重奏させるものです。

# 第4章 グラフィックの使い方

## 4-1 グラフィックの種類

MSX BASICは、4つの画面構成をもっており、目的に合わせて自由に選ぶことができます。この画面の指定はSCREEN文で行ないます。SCREENを切り換えると今までの画面は全て消えてしまいます。




## 4-2 SCREEN 0

このモードは最大40文字×24行のテキストモードです。SCREEN 0を実行した直後は、1行の表示文字が39文字にセットされています。(WIDTH文で最大40文字まで表示できます)
このモードでは、英字・数字以外の文字は、はっきりと表示されない場合がありますが、文字情報が多く表示できますので、リストを表示したりするのに便利です。
このモードでは、スプライト表示はできません。また、色の指定が2色(前景色と背景色)のみ有効になります。

```
10 SCREEN 0:WIDTH 40
20 KEY OFF
30 DEFINT A-Z
40 A=RND(-TIME)
50 PRINT "ｱﾙﾌｧﾍﾞｯﾄ ｳﾁｵﾄｼ ｹﾞｰﾑ"
60 PRINT
70 PRINT"A-H ﾉ ﾓｼﾞｦ ｳﾁｵﾄｼﾏｽ"
80 PRINT"ﾐｻｲﾙ ﾉ ﾊｯｼｬ ﾊ SPACE ｷｰ ﾃﾞｽ"
90 FOR T=1 TO 3000:NEXT T
100 CLS
110 LOCATE  8,0:PRINT"ABCDEFGFEDCBA"
```

```
120 LOCATE 29,0:PRINT"A:  1 POINT"
130 LOCATE 29,1:PRINT"B:  2 POINT"
140 LOCATE 29,2:PRINT"C:  3 POINT"
150 LOCATE 29,3:PRINT"D:  4 POINT"
160 LOCATE 29,4:PRINT"E:  5 POINT"
170 LOCATE 29,5:PRINT"F:  6 POINT"
180 LOCATE 29,6:PRINT"G:  7 POINT"
190 L=7:P=0:N=1:X=0
200 LOCATE 29,8:PRINT"POINT     0"
210 LOCATE 29,9:PRINT"LEFT     ";L
220 M1$="^"
230 M2$="X"
240 '
250 DIM PT(13)
260 FOR J=1 TO 13
270  READ PT(J)
280 NEXT J
290 '
300 DATA 1,2,3,4,5,6,7,6,5,4,3,2,1
310 '
320 LOCATE X,22:PRINT M1$;
330 LOCATE X,23:PRINT M2$;
340 A=STRIG(0)
350 IFNOT A THEN 490
360 FOR Y=22 TO 0 STEP -1
370   LOCATE X,Y:PRINT"^";
380   LOCATE X,Y+1:PRINT" ";
390 NEXT Y
400 LOCATE X,0:PRINT " ";
410 IF X<8 OR X>20 THEN 450
420 J=X-7
430 P=P+PT(J)
440 PT(J)=0
450 L=L-1
460 LOCATE 37,8:PRINT USING"##";P
470 LOCATE 37,9:PRINT USING"##";L
480 IF L=0 THEN 560
490 S=RND(1)*2
500 LOCATE X,22:PRINT " ";
510 LOCATE X,23:PRINT " ";
520 X=X+S*N
530 IF X>27 THEN N=-1:X=27
540 IF X<0 THEN N=1:X=0
550 GOTO 320
560 LOCATE 5,10:PRINT USING"POINT ###";P
570 LOCATE 5,12:PRINT"HIT ANY KEY!!";
580 IF INKEY$<>""THEN 580
590 A$=INPUT$(1)
```

画面上にAからGまでの文字がならんでいます。
この文字を打ち落とすゲームです。得点(ポイント)は画面右上にそれぞれ表示されています。ミサイルは7個まで、高得点をだすには、ポイントの高い文字をねらいます。
240、290、310行はアポストロフィーです。

## 4-3 SCREEN 1

このモードでは、ひらがな、グラフィック文字もはっきり表示することができます。画面に表示できる最大文字数は32文字×24行となります。またWIDTHで、横32文字に指定すると、画面の左端の文字が映らない場合があります。このようなことから、MSX BASICを作動したときはWIDTHが29文字になっています。

スプライト機能を使うことはできますが、画素(ドット)ごとの色を使うことはできません。色の指定は前景色、背景色、周辺色についてそれぞれ行なうことができます。

## 4-4 SCREEN 2

このモードは、高解像度表示モードと呼ばれ、たて192×横256ドットの表示ができます。また色の指定も各ドットごとに指定することができます。しかし、横8ドットに対して2色しか表現できません。したがって横8ドット中に2色を使っていて、3色目を指定しようとすると、すでに色をつけた2色のどちらかが3色目に変更されてしまいます。
このモードでは、スプライトもグラフィックコマンドも利用できますので、ゲームのプログラムや動画を作るのに利用します。
では、実際に順を追って説明していきましょう。

①背景の指定

画面上で背景の指定を行います。そのためには、まずSCREEN 2の画面にしたあとにLINE文やPSET文、PRINT文などで自由に絵を書きます。またDRAW文を使用することもできます。

②スプライトの定義をします

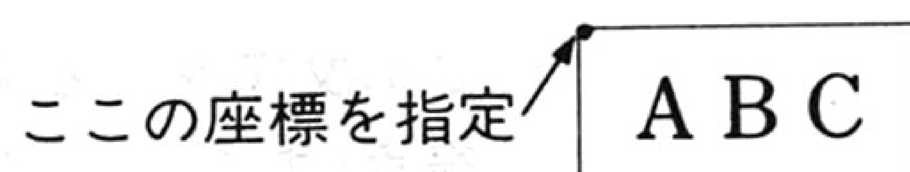
スプライトの定義は8×8ドット、または16×16ドットが指定できます。SPRITE＄を使いますので、「SPRITE＄の使い方」(202P) を参照してください。

③スプライトを動かす

スプライトを動かすには、スプライト座標のX座標とY座標の値を変更します。

④画面上に文字を出すには

テキスト画面とちがい、LOCATE文やPRINT文が使えません。ではどのようにすればいいでしょうか。
それにはまず表示する座標を指定します。この座標の最終参照点 (LP) の値を変更することでできます。いちばん簡単な方法は、PSETもしくはPRESETを使います。PSET、PRESETとも背景と同じ色になるようにします。座標は文字を8×8ドットのブロックとし、その左上の角の点を指定します。




次に指定した座標から文字を出すことにします。PRINT文は使用できませんので、OPEN文で〝GRP：″(グラフィック画面) をOPENし、〝GRP：″に対して出力します。その上からスペースを書いても消えません。〝GRP：″に対しての出力は下の文字を構成するドットをそのままにして、新たな文字を構成するドットを付加するだけだからです。したがって文字を消すいちばん簡単な方法は、下の色(背景色)で塗ってしまうことです。

⑤今までの動きをまとめたプログラムを作ってみましょう。

- SCREEN文で画面モードを設定します。
- 背景を描きます。
- ヒヨコ (スプライトで定義) を表示します。
- 文字を出力します。
- ヒヨコを動かします。
- 文字を出力します。

```
10 COLOR 15,5,5
20 SCREEN 2,2
30 CIRCLE (128,375),306,12
40 PAINT (128,191),12
50 CIRCLE (200,15),30,15,,,.3
60 PAINT (200,15),15
70 CIRCLE (215,30),25,15,,,.3
80 PAINT (215,30),15
90 FOR J=1 TO 16
100 READ D$
110 A$=A$+CHR$(VAL("&B"+LEFT$(D$,8)))
120 B$=B$+CHR$(VAL("&B"+RIGHT$(D$,8)))
130 NEXT J
140 SPRITE$(0)=A$+B$
150 PUT SPRITE 0,(230,130),9,0
160 FOR T=1 TO 500:NEXT T
170 PRESET (10,10)
180 OPEN "GRP:" FOR OUTPUT AS #1
190 PRINT #1,"ﾋﾖｺｦ ｳｺﾞｶｼﾏｼｮｳ"
200 CLOSE #1
210 X=220
220 PUT SPRITE 0,(X,130),9,0
230 FOR T=1 TO 500:NEXT T
240 X=X-10
250 IF X=0 THEN 270
260 GOTO 220
270 X=X+10
280 IF X=240 THEN 320
290 PUT SPRITE 0,(X,130),9,0
300 TIME=0
310 IF TIME=20 THEN 270 ELSE 310
320 LINE (10,10)-(130,18),5,BF
330 PRESET (10,0)
340 OPEN "GRP:" FOR OUTPUT AS #1
350 PRINT #1,"  ﾔﾒﾙﾄｷﾊ ｷｰﾎﾞｰﾄﾞｦ"
360 PRINT #1,"   ｵｼﾃｸﾀﾞｻｲ"
370 CLOSE #1
380 A$=INPUT$(1)
390 END
400 DATA 0000000110000000
410 DATA 0000000110000000
420 DATA 0000111111110000
430 DATA 0001111111111000
440 DATA 0001111111111000
450 DATA 0001110110111000
460 DATA 0001110110111000
470 DATA 1111111111111111
480 DATA 1111111001111111
490 DATA 0111110000111110
500 DATA 0111111001111110
510 DATA 0011111111111100
520 DATA 0000111111110000
530 DATA 0000010110100000
540 DATA 0000010000100000
550 DATA 0011111001111100
```

何かキーを押すことでプログラムは終わります。

# 第5章 ジョイスティック、ジョイパッド

## 5-1 ジョイスティックとは

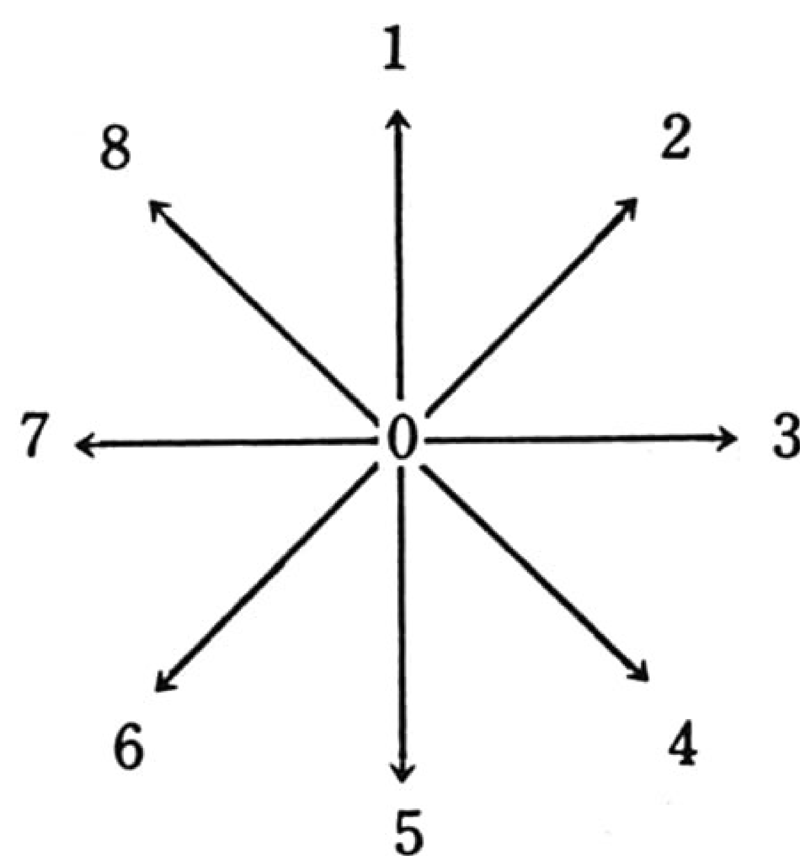
ゲームをするときに、誰でも使うジョイスティックは、操作が簡単で思わずゲームに夢中になってしまうほどです。
このジョイスティックは、MSXではカーソルキーでも代用できます。また、MX-101ではジョイパッドと呼ばれる独自の操作部がついていますので、ジョイスティックを購入しなくても十分ゲームを楽しむことができます。

## 5-2 ジョイスティック番号の割り付け

ジョイスティックの番号は0がキーボードのカーソル移動キー、1がポートにつながったジョイスティック、もしくはジョイパッドになります。ポート2につながったジョイスティックの番号は2となります。

## 5-3 ジョイスティックの方向

ジョイスティックがどちらに倒されているかはSTICK関数で知ることができます。ジョイスティックの倒れている方向によって下図の値が得られます。どちらの方向にも倒れていないときは0が得られます。



## 5-4 ジョイスティックのボタン

ジョイスティックにはトリガーボタンが2つついています。このボタンが押されているかいないかは、STRIG関数でその状態を知ることができます。カーソルを代用するジョイスティック0はスペースキーがトリガーボタンのひとつになります。2番目のトリガーボタンは存在しません。

| ジョイスティック番号 | ポート番号 | トリガーボタン |
|---|---|---|
| 0 | スペースキー | |
| 1 | 1 | 1 |
| 2 | 2 | 1 |
| 3 | 1 | 2 |
| 4 | 2 | 2 |

## 5-5 簡単なプログラムを作ってみましょう

ここでは、ジョイパッドを利用して簡単なプログラムを作ってみます。ジョイパッドを動かすことにより、動かした方向に線を描くことができます。トリガーボタンの1を押せば画面が消え、トリガーボタンの2を押せば、プログラムが終わります。

```
10 COLOR 15,4,7:SCREEN2
20 DEFINT A-Z
30 X=256/2
40 Y=192/2
50 N=STICK(1)
60 IF N=1 THEN Y=Y-1
70 IF N=2 THEN X=X+1:Y=Y-1
80 IF N=3 THEN X=X+1
90 IF N=4 THEN X=X+1:Y=Y+1
100 IF N=5 THEN Y=Y+1
110 IF N=6 THEN X=X-1:Y=Y+1
120 IF N=7 THEN X=X-1
130 IF N=8 THEN X=X-1:Y=Y-1
140 IF X<0 THEN X=0
150 IF Y<0 THEN Y=0
160 IF X>255 THEN X=255
170 IF Y>191 THEN Y=191
180 PSET (X,Y),6
190 IF STRIG(1) THEN 10
200 IF STRIG(3) THEN 300
210 GOTO 50
300 COLOR 15,4,7
310 END
```

# 第6章 カセットファイルの使い方

## 6-1 ファイルとは?

コンピュータは絵を描いたり、音楽を演奏させる以外に、“データ処理”を行なうことができます。この“データ処理”は、計算処理であったり、ファイル処理であったりします。ファイル処理というと、ちょっとむずかしく聞こえますが、住所録などを記録するのもひとつのファイルです。
では、その“住所録ファイル”を例にいろいろ考えてみましょう。住所録の項目は、名前、住所、年令の3つとします。
1人分の名前のデータ、住所のデータ、年令のデータを合わせて“レコード”と呼びます。つまり1人分1人分がそれぞれ1レコードであり、そのレコードが集まってひとつのファイルを作っています。

| 名　　前 | 住　　所 | 年令 |
|---|---|---|
| カシオ　タロウ | トウキョウト　シンジュクク2-6 | 27 |
| アサヒ　ジロウ | ホッカイドウ　アサヒカワシ8-1 | 23 |
| フクオカ　サブロウ | フクオカケン　フクオカシ　ハカタク2-1 | 21 |

## 6-2 ファイルを作るには

ファイルを作るには、ファイルを作る前にファイルをOPENします。このOPENというのは、これからファイルを作成する前準備として考えてください。このOPEN文で、出力はカセットテープでファイル名をTESTとすることを宣言します。

OPEN "CAS：TEST" FOR OUTPUT AS＃1

このファイルをファイル番号1とします。またメモリー内容をテープへ書き出すので出力モードになります。では、実際のプログラムを打ち込み実行してみてください。データレコーダーにテープをセットし、録音ボタンを押してからRUN ⏎と操作します。

```
10 OPEN "CAS:TEST" FOR OUTPUT AS #1
20 NA$=""
30 INPUT"ﾅﾏｴ ﾊ(ｵﾜﾘﾊ､RETURN)";NA$
40 IF NA$="" THEN 110
50 PRINT #1,NA$
60 INPUT"ｼﾞｭｳｼｮ ";AD$
70 PRINT #1,AD$
80 INPUT"ﾈﾝﾚｲ ";NE%
90 PRINT #1,NE%
100 GOTO 20
110 PRINT #1,"0"
120 CLOSE
130 END
```

リモート端子のないテープレコーダーのときは 246 P 参照

PRINT＃1という命令はデータファイルに対して出力する命令です。＃1は1番のファイル、(10行で＃1をカセットに定義しているので) つまりカセットに対するファイルを意味します。データの最後は“0”を書き出し、CLOSE処理します。CLOSEというのは、もうデータがなく、ファイルの書き出しの終わりを意味します。以降PRINT＃1を実行できません。

## 6-3 ファイルを読み込むには

下のプログラムを入力した後に、さきほど作った“住所録”のテープを巻きもどし、データレコーダーを再生してください。まずOPEN文でカセットファイルをOPENします。

OPEN "CAS：TEST" FOR INPUT AS＃1

これは、ファイル番号1として、カセットファイルを定義します。もちろん入力モードにします。名前のデータを読み込んだあとで“0”かどうかをチェックし、0ならば終わりにします。データを読み込むときはLINE INPUT＃またはINPUT＃を使います。文字列は、LINE INPUT＃を使うことで、カンマ(，)の入力もできますので、名前や住所中に，を含むことができます。読み込む変数は書き出しに使用した変数名と異ってもかまいませんが型は一致していなければなりません。

```
10 OPEN "CAS:TEST" FOR INPUT  AS #1
20 LINE INPUT #1,NA$
30 IF NA$="0" THEN 80
40 PRINT"ﾅﾏｴ ﾊ ";NA$
50 LINE INPUT #1,AD$:PRINT"ｼﾞｭｳｼｮ ";AD$
60 INPUT #1,NE%:PRINT"ﾈﾝﾚｲ ";NE%
70 GOTO 20
80 CLOSE
90 END
```

リモート端子のないテープレコーダーのときは 246 P 参照

## テクニック

リモート端子のあるデータレコーダーからデータを読み込むときは、再生ボタンを押してからプログラムを実行すれば、自動的にデータレコーダーからデータが読み込まれます。しかし、リモート端子のないデータレコーダーでは、プログラム実行前に再生ボタンを押せば、その時点で動作してしまいます。

MX-101ではカセットファイルに対してのOPEN文、CLOSE文を実行するとMX-101本体の中にあるリレー(スイッチ)がカチッと鳴ります。この音を目安に再生ボタン、停止ボタンを押せばよいのですが、聞きもらす心配もあります。そこで、PRINT文で画面にメッセージを出すようにすると確実です。メッセージのかわりに音楽を利用しても楽しいと思います。

①ファイルを作るには……

追加

```
5 PRINT"ﾚｺｰﾀﾞｰ ｦ ﾛｸｵﾝ ｼﾞｮｳﾀｲ ﾆ"
6 PRINT"ﾘﾀｰﾝ ｷｰ ｦ ｵｽ"
7 A$=INPUT$(1):IF A$<>CHR$(13) THEN 7
125 PRINT"ﾚｺｰﾀﾞｰ ｦ ﾄﾒﾙ"
```

変更例

```
5 PRINT"ﾚｺｰﾀﾞｰ ｦ ﾛｸｵﾝ ｼﾞｮｳﾀｲ ﾆ"
6 PRINT"ﾘﾀｰﾝ ｷｰ ｦ ｵｽ"
7 A$=INPUT$(1):IF A$<>CHR$(13) THEN 7
10 OPEN "CAS:TEST" FOR OUTPUT AS #1
20 NA$=""
30 INPUT"ﾅﾏｴ ﾊ (ｵﾜﾘﾊ､RETURN)";NA$
40 IF NA$="" THEN 110
50 PRINT #1,NA$
60 INPUT"ｼﾞｭｳｼｮ ";AD$
70 PRINT #1,AD$
80 INPUT"ﾈﾝﾚｲ ";NE%
90 PRINT #1,NE%
100 GOTO 20
110 PRINT #1,"0"
120 CLOSE
125 PRINT"ﾚｺｰﾀﾞｰ ｦ ﾄﾒﾙ"
130 END
```

②ファイルを読み込むには……

追加

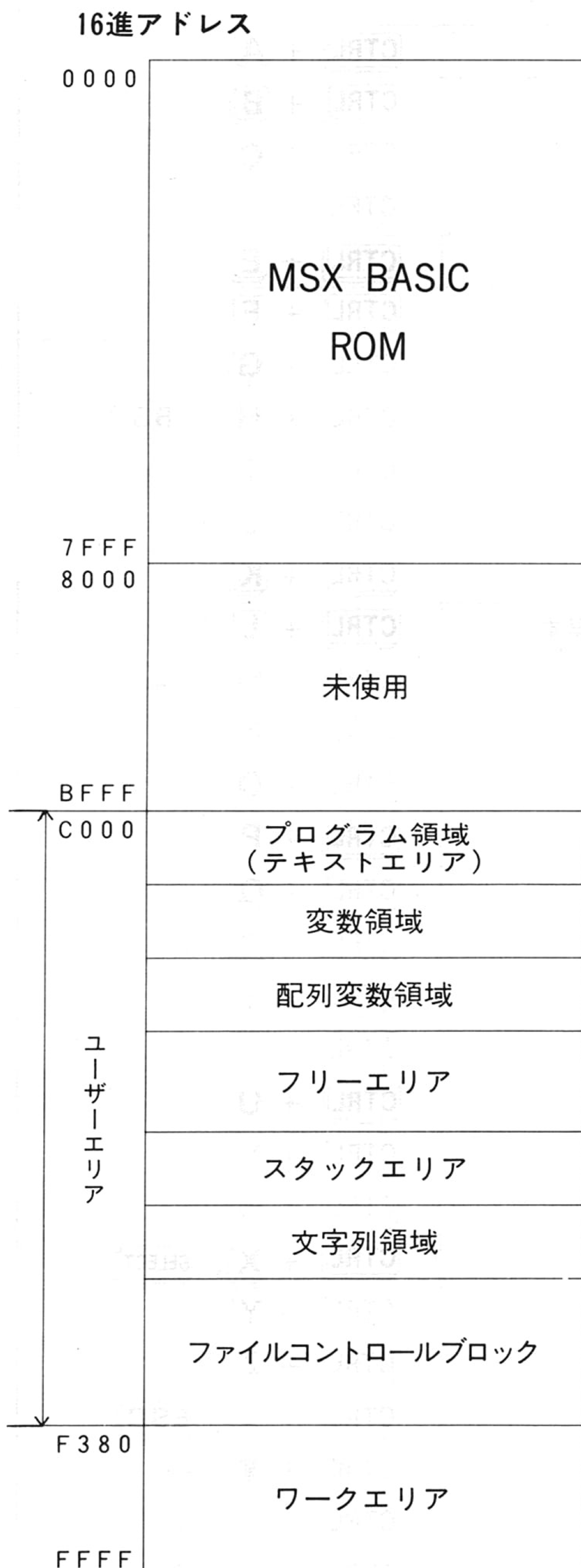
```
5 PRINT "ﾚｺｰﾀﾞｰ ｦ ｻｲｾｲ ﾆ"
85 PRINT "ﾚｺｰﾀﾞｰ ｦ ﾄﾒﾙ"
```

変更例

```
5 PRINT "ﾚｺｰﾀﾞｰ ｦ ｻｲｾｲ ﾆ"
10 OPEN "CAS:TEST" FOR INPUT AS #1
20 LINE INPUT #1,NA$
30 IF NA$="0" THEN 80
40 PRINT "ﾅﾏｴ ﾊ ";NA$
50 LINE INPUT #1,AD$:PRINT "ｼﾞｭｳｼｮ ";AD$
60 INPUT #1,NE%:PRINT "ﾈﾝﾚｲ ";NE%
70 GOTO 20
80 CLOSE
85 PRINT "ﾚｺｰﾀﾞｰ ｦ ﾄﾒﾙ"
90 END
```

# 付　録

## (1)メモリーマップ




**●プログラム領域(テキストエリア)**

行番号を付けたプログラムが格納される領域です。

**●変数領域**

変数のための領域です。文字変数の場合は、用意された文字列へのポインタ（ストリングディスクリプタ）が格納されます。

**●配列変数領域**

配列変数のための領域です。配列変数が文字型ならば文字列領域へ用意された文字列へのポインタが格納されます。この領域はDIM文を実行したり、添字が10以下の配列をDIM文無しで使ったときに確保されます。

**●フリーエリア**

未使用領域です。(このエリアの大きさは、ユーザエリアの大きさから文字領域、スタック領域、変数領域、プログラム領域を差し引いたもので、FRE関数で求められます。)

**●スタックエリア**

FOR～NEXT文やGOSUB文などを実行するときにBASICが戻りアドレスを退避する領域です。

**●文字列領域**

文字変数や配列変数に含まれる文字列を格納する領域です。この領域の大きさはCLEAR文で指定した大きさです。指定がなければ200バイトの領域が確保されます。

**●ファイルコントロールブロック**

ファイルの入出力時に使用する領域です。MAXFILES文で指定した数に応じて確保されます。上限アドレスをCLEAR文でF380(16進)以下に設定できますので、ワークエリアとの間に機械語ルーチンなどのために、ユーザーが自由に使うことができる領域を設けることもできます。

**●ワークエリア**

BASICが使用する領域です。

## (2)コントロールコード表

| コード(10進) | コード(16進) | 機　　　　能 | 対応キー |
|---|---|---|---|
| 0 | 00 | | |
| 1 | 01 | グラフィックキャラクターの入出力時のヘッダ | CTRL + A |
| 2 | 02 | カーソルを直前の語の先頭へ移動 | CTRL + B |
| 3 | 03 | 入力待ち状態を終了する | CTRL + C |
| 4 | 04 | | CTRL + D |
| 5 | 05 | カーソル以下を削除 | CTRL + E |
| 6 | 06 | カーソルを次の語の先頭へ移動 | CTRL + F |
| 7 | 07 | スピーカを鳴らす(BEEP文と同じ) | CTRL + G |
| 8 | 08 | カーソルの1つ前の文字を削除する | CTRL + H, BS |
| 9 | 09 | 次の水平タブ位置へ移動 | CTRL + I, TAB |
| 10 | 0A | 行送り(ラインフィード) | CTRL + J |
| 11 | 0B | カーソルをホームポジション(左上)に戻す | CTRL + K |
| 12 | 0C | 画面をクリアし、カーソルをホームポジションに戻す | CTRL + L |
| 13 | 0D | カーソルを左端に戻す(キャリッジリターン) | CTRL + M, ↵ |
| 14 | 0E | カーソルを行末へ移動 | CTRL + N |
| 15 | 0F | | CTRL + O |
| 16 | 10 | | CTRL + P |
| 17 | 11 | | CTRL + Q |
| 18 | 12 | 挿入モードのON/OFFスイッチ | CTRL + R |
| 19 | 13 | | CTRL + S |
| 20 | 14 | | CTRL + T |
| 21 | 15 | 一行を画面から削除 | CTRL + U |
| 22 | 16 | | CTRL + V |
| 23 | 17 | | CTRL + W |
| 24 | 18 | | CTRL + X, SELECT |
| 25 | 19 | | CTRL + Y |
| 26 | 1A | | CTRL + Z |
| 27 | 1B | | CTRL + [, ESC |
| 28 | 1C | カーソルを右へ移動 | CTRL + ¥, ➡ |
| 29 | 1D | カーソルを左へ移動 | CTRL + ], ⬅ |
| 30 | 1E | カーソルを上へ移動 | CTRL + ^, ⬆ |
| 31 | 1F | カーソルを下へ移動 | CTRL + SHIFT → ?/_, ⬇ |
| 127 | 7F | カーソルの指す文字を削除 | CTRL + DEL |

## (3)キャラクター・コード表

| コード(10進) | コード(16進) | CHR | コード(10進) | コード(16進) | CHR | コード(10進) | コード(16進) | CHR | コード(10進) | コード(16進) | CHR | コード(10進) | コード(16進) | CHR | コード(10進) | コード(16進) | CHR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | ↑ | 43 | 2B | + | 86 | 56 | V | 129 | 81 | ♥ | 172 | AC | ャ | 215 | D7 | ラ |
| 1 | 01 | | 44 | 2C | , | 87 | 57 | W | 130 | 82 | ♣ | 173 | AD | ュ | 216 | D8 | リ |
| 2 | 02 | | 45 | 2D | − | 88 | 58 | X | 131 | 83 | ◆ | 174 | AE | ョ | 217 | D9 | ル |
| 3 | 03 | | 46 | 2E | . | 89 | 59 | Y | 132 | 84 | ○ | 175 | AF | ッ | 218 | DA | レ |
| 4 | 04 | | 47 | 2F | ／ | 90 | 5A | Z | 133 | 85 | ● | 176 | B0 | ー | 219 | DB | ロ |
| 5 | 05 | | 48 | 30 | 0 | 91 | 5B | [ | 134 | 86 | を | 177 | B1 | ア | 220 | DC | ワ |
| 6 | 06 | | 49 | 31 | 1 | 92 | 5C | ¥ | 135 | 87 | ぁ | 178 | B2 | イ | 221 | DD | ン |
| 7 | 07 | | 50 | 32 | 2 | 93 | 5D | ] | 136 | 88 | ぃ | 179 | B3 | ウ | 222 | DE | ゛ |
| 8 | 08 | | 51 | 33 | 3 | 94 | 5E | ∧ | 137 | 89 | ぅ | 180 | B4 | エ | 223 | DF | ゜ |
| 9 | 09 | | 52 | 34 | 4 | 95 | 5F | — | 138 | 8A | ぇ | 181 | B5 | オ | 224 | E0 | た |
| 10 | 0A | | 53 | 35 | 5 | 96 | 60 | ` | 139 | 8B | ぉ | 182 | B6 | カ | 225 | E1 | ち |
| 11 | 0B | コントロールキャラクター | 54 | 36 | 6 | 97 | 61 | a | 140 | 8C | ゃ | 183 | B7 | キ | 226 | E2 | つ |
| 12 | 0C | | 55 | 37 | 7 | 98 | 62 | b | 141 | 8D | ゅ | 184 | B8 | ク | 227 | E3 | て |
| 13 | 0D | | 56 | 38 | 8 | 99 | 63 | c | 142 | 8E | ょ | 185 | B9 | ケ | 228 | E4 | と |
| 14 | 0E | | 57 | 39 | 9 | 100 | 64 | d | 143 | 8F | っ | 186 | BA | コ | 229 | E5 | な |
| 15 | 0F | | 58 | 3A | : | 101 | 65 | e | 144 | 90 | | 187 | BB | サ | 230 | E6 | に |
| 16 | 10 | | 59 | 3B | ; | 102 | 66 | f | 145 | 91 | あ | 188 | BC | シ | 231 | E7 | ぬ |
| 17 | 11 | | 60 | 3C | < | 103 | 67 | g | 146 | 92 | い | 189 | BD | ス | 232 | E8 | ね |
| 18 | 12 | | 61 | 3D | = | 104 | 68 | h | 147 | 93 | う | 190 | BE | セ | 233 | E9 | の |
| 19 | 13 | | 62 | 3E | > | 105 | 69 | i | 148 | 94 | え | 191 | BF | ソ | 234 | EA | は |
| 20 | 14 | | 63 | 3F | ? | 106 | 6A | j | 149 | 95 | お | 192 | C0 | タ | 235 | EB | ひ |
| 21 | 15 | | 64 | 40 | @ | 107 | 6B | k | 150 | 96 | か | 193 | C1 | チ | 236 | EC | ふ |
| 22 | 16 | | 65 | 41 | A | 108 | 6C | l | 151 | 97 | き | 194 | C2 | ツ | 237 | ED | へ |
| 23 | 17 | | 66 | 42 | B | 109 | 6D | m | 152 | 98 | く | 195 | C3 | テ | 238 | EE | ほ |
| 24 | 18 | | 67 | 43 | C | 110 | 6E | n | 153 | 99 | け | 196 | C4 | ト | 239 | EF | ま |
| 25 | 19 | | 68 | 44 | D | 111 | 6F | o | 154 | 9A | こ | 197 | C5 | ナ | 240 | F0 | み |
| 26 | 1A | | 69 | 45 | E | 112 | 70 | p | 155 | 9B | さ | 198 | C6 | ニ | 241 | F1 | む |
| 27 | 1B | | 70 | 46 | F | 113 | 71 | q | 156 | 9C | し | 199 | C7 | ヌ | 242 | F2 | め |
| 28 | 1C | | 71 | 47 | G | 114 | 72 | r | 157 | 9D | す | 200 | C8 | ネ | 243 | F3 | も |
| 29 | 1D | | 72 | 48 | H | 115 | 73 | s | 158 | 9E | せ | 201 | C9 | ノ | 244 | F4 | や |
| 30 | 1E | | 73 | 49 | I | 116 | 74 | t | 159 | 9F | そ | 202 | CA | ハ | 245 | F5 | ゆ |
| 31 | 1F | ↓ | 74 | 4A | J | 117 | 75 | u | 160 | A0 | | 203 | CB | ヒ | 246 | F6 | よ |
| 32 | 20 | | 75 | 4B | K | 118 | 76 | v | 161 | A1 | 。 | 204 | CC | フ | 247 | F7 | ら |
| 33 | 21 | ! | 76 | 4C | L | 119 | 77 | w | 162 | A2 | 「 | 205 | CD | ヘ | 248 | F8 | り |
| 34 | 22 | " | 77 | 4D | M | 120 | 78 | x | 163 | A3 | 」 | 206 | CE | ホ | 249 | F9 | る |
| 35 | 23 | # | 78 | 4E | N | 121 | 79 | y | 164 | A4 | 、 | 207 | CF | マ | 250 | FA | れ |
| 36 | 24 | $ | 79 | 4F | O | 122 | 7A | z | 165 | A5 | ・ | 208 | D0 | ミ | 251 | FB | ろ |
| 37 | 25 | % | 80 | 50 | P | 123 | 7B | { | 166 | A6 | ヲ | 209 | D1 | ム | 252 | FC | わ |
| 38 | 26 | & | 81 | 51 | Q | 124 | 7C | \| | 167 | A7 | ァ | 210 | D2 | メ | 253 | FD | ん |
| 39 | 27 | ' | 82 | 52 | R | 125 | 7D | } | 168 | A8 | ィ | 211 | D3 | モ | 254 | FE | |
| 40 | 28 | ( | 83 | 53 | S | 126 | 7E | ~ | 169 | A9 | ゥ | 212 | D4 | ヤ | 255 | FF | |
| 41 | 29 | ) | 84 | 54 | T | 127 | 7F | | 170 | AA | ェ | 213 | D5 | ユ | | | |
| 42 | 2A | * | 85 | 55 | U | 128 | 80 | ♠ | 171 | AB | ォ | 214 | D6 | ヨ | | | |

### ●グラフィックキャラクターコード表

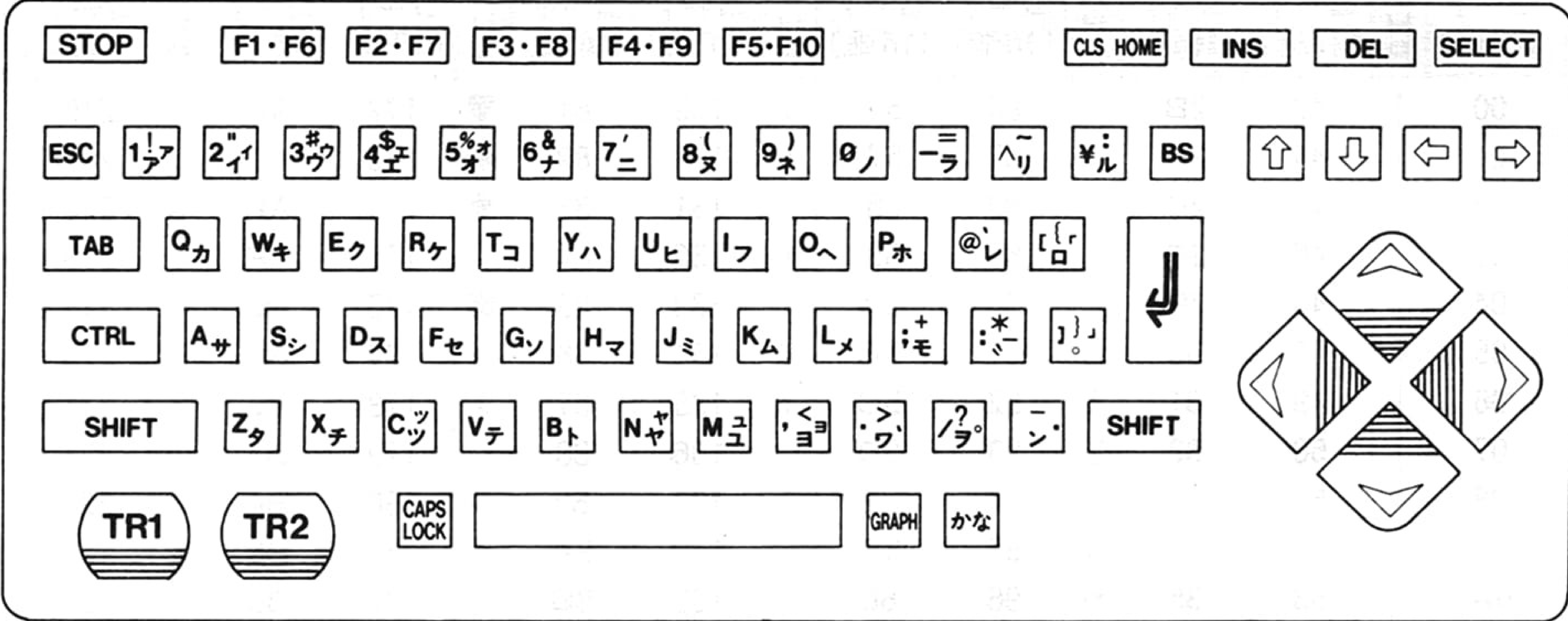
10進数のコード0～31(16進数では00～1F)はコントロールコードです。

グラフィック文字の出力は、PRINT　CHR＄(1)+CHR＄(表示文字コード+64)と行ないます。

月の表示→PRINT　CHR＄(1)+CHR＄(65)(月の表示文字コードは1)

| コード(10進) | コード(16進) | キャラクター | コード(10進) | コード(16進) | キャラクター | コード(10進) | コード(16進) | キャラクター | コード(10進) | コード(16進) | キャラクター | コード(10進) | コード(16進) | キャラクター | コード(10進) | コード(16進) | キャラクター |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | | 6 | 06 | 土 | 12 | 0C | 秒 | 18 | 12 | ┬ | 24 | 18 | ┌ | 30 | 1E | 中 |
| 1 | 01 | 月 | 7 | 07 | 日 | 13 | 0D | 百 | 19 | 13 | ┤ | 25 | 19 | ┐ | 31 | 1F | 小 |
| 2 | 02 | 火 | 8 | 08 | 年 | 14 | 0E | 千 | 20 | 14 | ├ | 26 | 1A | └ | | | |
| 3 | 03 | 水 | 9 | 09 | 円 | 15 | 0F | 万 | 21 | 15 | ┼ | 27 | 1B | ┘ | | | |
| 4 | 04 | 木 | 10 | 0A | 時 | 16 | 10 | π | 22 | 16 | │ | 28 | 1C | ╳ | | | |
| 5 | 05 | 金 | 11 | 0A | 分 | 17 | 11 | ┴ | 23 | 17 | ─ | 29 | 1D | 大 | | | |

## ⑷キー配列




## ⑸誘導関数

| 数式 | 求める関数 | 誘導式 |
|---|---|---|
| sec | セカント | SEC(x) = 1/COS(x) |
| cosec | コセカント | CSC(x ) = 1/SIN(x) |
| cot | コタンジェント | COT(x) = 1/TAN(x) |
| $\sin^{-1}$ | アークサイン | ARCSIN(x) = ATN(x/SQR(−x*x+1)) |
| $\cos^{-1}$ | アークコサイン | ARCCOS=−ATN(x/SQR(−x*x+1))+1.57.08 |
| $\sec^{-1}$ | アークセカント | ARCSEC(x)=ATN(x*x−1))+(SGN(x)−1)*1.5708 |
| $\mathrm{cosec}^{-1}$ | アークコセカント | ARCCSC(x)=ATN(1/SQR(x*x−1))+(SGN(x)−1)*1.5708 |
| $\cot^{-1}$ | アークコタンジェント | ARCCOT(x) = −ATN(x) + 1.5708 |
| sinh | ハイパボリック・サイン | SINH(x) = (EXP(x)−EXP(−x))/2 |
| cosh | ハイパボリック・コサイン | COSH(x) = (EXP(x)+EXP(−x))/2 |
| tanh | ハイパボリック・タンジェント | TANH(x)=−EXP(−x)/(EXP(x)+EXP(−x))*2+1 |
| sech | ハイパボリック・セカント | SECH(x) = 2/(EXP(x)+EXP(−x)) |
| cosech | ハイパボリック・コセカント | CSCH(x) = 2/(EXP(x)−EXP(−x)) |
| coth | ハイパボリック・コタンジェント | COTH(x)=EXP(−x)/(EXP(x)−EXP(−x))*2+1 |
| $\sinh^{-1}$ | ハイパボリック・アークサイン | ARCSINH(x) = LOG(x+SQR(x*x+1)) |
| $\cosh^{-1}$ | ハイパボリック・アークコサイン | ARCCOSH(x) = LOG(x+SQR(x*x+1)) |
| $\tanh^{-1}$ | ハイパボリック・アークタンジェント | ARCTANH(x) = LOG((1+x)/(1−x))/2 |
| $\mathrm{sech}^{-1}$ | ハイパボリック・アークセカント | ARCSECH(x)=LOG((SQR(−x*x+1)+1)/x) |
| $\mathrm{cosech}^{-1}$ | ハイパボリック・アークコセカント | ARCCSCH(x)=LOG((SGN(x)*SQR(x*x+1)+1)/x) |
| $\coth^{-1}$ | ハイパボリック・アークコタンジェント | ARCCOTH(x) = LOG((x+1)/(x−1))/2 |
| $\mathrm{Log}_{10}$ | 常用対数 | $\mathrm{LOG}_{10}$(x)=LOG(x)/LOG(10) |

## (6) MSX-BASICの予約語

| | | | |
|---|---|---|---|
| ABS | DSKI$ | LOC | RESUME |
| AND | DSKO$ | LOCATE | RETURN |
| ASC | ELSE | LOF | RIGHT$ |
| ATN | END | LOG | RND |
| ATTR$ | EOF | LPOS | RSET |
| AUTO | EQV | LPRINT | RUN |
| BASE | ERASE | LSET | SAVE |
| BEEP | ERL | MAX | SCREEN |
| BIN$ | ERR | MERGE | SET |
| BLOAD | ERROR | MID$ | SGN |
| BSAVE | EXP | MKD$ | SIN |
| CALL | FIELD | MKI$ | SOUND |
| CDBL | FILES | MKS$ | SPACE$ |
| CHR$ | FIX | MOD | SPC( |
| CINT | FN | MOTOR | SPRITE |
| CIRCLE | FOR | NAME | SQR |
| CLEAR | FPOS | NEW | STEP |
| CLOAD | FRE | NEXT | STICK |
| CLOSE | GET | NOT | STOP |
| CLS | GO TO | OCT$ | STR$ |
| CMD | GOSUB | OFF | STRIG |
| COLOR | GOTO | ON | STRING$ |
| CONT | HEX$ | OPEN | SWAP |
| COPY | IF | OR | TAB( |
| COS | IMP | OUT | TAN |
| CSAVE | INKEY$ | PAD | THEN |
| CSNG | INP | PAINT | TIME |
| CSRLIN | INPUT | PDL | TO |
| CVD | INSTR | PEEK | TROFF |
| CVI | INT | PLAY | TRON |
| CVS | IPL | POINT | USING |
| DATA | KEY | POKE | USR |
| DEF | KILL | POS | VAL |
| DEFDBL | LEFT$ | PRESET | VARPTR |
| DEFINT | LEN | PRINT | VDP |
| DEFSNG | LET | PSET | VPEEK |
| DEFSTR | LFILES | PUT | VPOKE |
| DELETE | LINE | READ | WAIT |
| DIM | LIST | REM | WIDTH |
| DRAW | LLIST | RENUM | XOR |
| DSKF | LOAD | RESTORE | |

※上記予約語は変数として使用できません。

## (7)エラーメッセージ一覧表

| メッセージ | エラーコード | 説明 | 対処方法 |
|---|---|---|---|
| バッド ファイル ネーム<br>Bad file name | 56 | ファイル名の指定に誤まりがある | ファイル名を検討してみます。 |
| バッド ファイル ナンバー<br>Bad file number | 52 | ①MAXFILES文で指定した範囲を越えたファイル番号を使おうとした。<br>②OPENしていないファイルを使おうとした。 | ①ファイル番号を確かめ、必要ならばMAXFILES文の値を変更する。<br>②ファイルをOPENする。 |
| キャント コンティニュー<br>Can't CONTINUE | 17 | CONT命令でプログラムが実行できない。エラーで停止したプログラムやプログラムがメモリー上にないのにCONTを実行した。 | プログラムを修正するとCONTできなくなります。RUNかGOTOで実行する。 |
| ディバイス アイオー エラー<br>Device I/O error | 19 | カセットテープの読み込みエラーやプリンタ、ディスプレイのエラー | カセットテープの再生レベルを変えたりプリンタの接続を確認します。または、ハードの故障についても確認します。 |
| ダイレクト ステートメント イン ファイル<br>Direct statement in file | 57 | ①アスキー形式のプログラムをロード中に行番号のない行をロードしようとした。<br>②データファイルをロードした。 | ロードしようとしたファイルの内容を確認します。 |
| ディヴィジョン バイ ゼロ<br>Division by zero | 11 | ①ゼロで割算した。<br>②負の指数のべき乗の結果がゼロになった。 | 計算式もしくは、変数の内容を確認します。 |
| フィールド オーバーフロー<br>FIELD overflow | 50 | ①OPEN文で指定したランダムファイルのレコード長(合計256バイト)を越えた。<br>②ランダムファイルに対してシーケンシャル入出力中にI/Oバッファがあふれた。 | FIELD文を確認します。 |
| ファイル オルレディ オープン<br>File already open | 54 | すでにOPEN中のファイルを再度OPENしたり、KILLしたりしようとした。 | 一担ファイルをCLOSE処理します。 |
| ファイル ナット ファウンド<br>File not found | 53 | LOAD、KILL、OPENなどで、存在しないファイル名を指定した。 | ファイル名を確認します。 |
| ファイル ナット オープン<br>File not OPEN | 59 | OPENされていないファイルに対して、PRINT#やINPUT#を行なおうとした。 | ファイルを一担OPENします。 |

| メッセージ | エラーコード | 説　　明 | 対処方法 |
|---|---|---|---|
| イリーガル　ダイレクト<br>Illegal direct | 12 | ダイレクトモードで使えないステートメントをダイレクトモードで実行しようとした。 | ダイレクトモードでは、ステートメントの使用に注意します。 |
| イリーガル ファンクション コール<br>Illegal function call | 5 | ①許された範囲外の値の引数を使った。<br>②ステートメントや関数の使い方が誤っている。 | 引数の内容を確認し、その内容がなぜそのようになったかを検討します。 |
| インプット パスト　エンド<br>Input past end | 55 | ファイルのレコードをすべて読み終ったのに、INPUT文をさらに実行しようとした。また、空のファイルに対してINPUTを実行した。 | なぜそのようになったかを検討し、読み込みをしないようにします。 |
| インターナル　エラー<br>Internal error | 51 | MSX BASIC内部のエラー。 | 通常はこのエラーは発生しません。 |
| ライン バッファー オーバーフロー<br>Line buffer overflow | 25 | プログラムの１行の長さの許されている範囲(１行255文字)を越えた。 | １行の長さを短かくします。 |
| ミッシング　オペランド<br>Missing operand | 24 | 式の記述が不完全 | 式の内容を確認します。 |
| ネクスト　ウィズアウト　フォー<br>NEXT without FOR | 1 | NEXT文の変数がFOR文と正しく対応していない。 | FORとNEXTの関係を調べてみます。 |
| ノー　リジュウム<br>No RESUME | 21 | ON ERROR GOTO文のエラーによる割り込み処理ルーチンの最後にRESUME文もEND文も存在しない。 | エラー処理ルーチンの最後はRESUME文にします。 |
| アウト オブ データ<br>Out of DATA | 4 | プログラムにREADできるデータがない。 | DATA文のデータの個数とREADを実行する回数を確認してみます。 |
| アウト オブ　メモリー<br>Out of memory | 7 | ①プログラムが大きすぎる。<br>②プログラムで使うファイルの数が多い。<br>③変数が多すぎメモリーがたりなくなった。<br>④式の表現が複雑すぎる。 | ①プログラムを短かくする。<br>②増設RAMカートリッジを接続しRAM容量をふやす。 |
| アウト オブ ストリング スペース<br>Out of string space | 14 | 文字変数領域の上限を越えて、文字列を使おうとした。<br>(文字変数用のメモリーがたりなくなった) | CLEAR文で使用できる文字変数領域を増やしたり、文字列の取扱いを少なくします。 |
| オーバー フロー<br>Overflow | 6 | 計算の結果が、BASICで許された範囲を越えた。 | 変数の型をかえます。取扱う数値を検討します。 |
| リディメンションド　アレイ<br>Redimensioned array | 10 | すでに宣言した配列変数に対してさらに宣言しようとした。DIM文を省略した配列(省略値10)に対してDIM文を定義した。 | 再宣言をしないようにするか、ERASEで一担配列変数を削除します。 |

| メッセージ | エラーコード | 説明 | 対処方法 |
|---|---|---|---|
| リジュウム ウィズアウト エラー<br>RESUME without error | 22 | エラーが発生していない状態でRESUMEを実行した。 | プログラムの流れを確認します。 |
| リターン ウィズアウト ゴーサブ<br>RETURN without GOSUB | 3 | RETURN文とGOSUBが正しく対応していない。 | プログラムの流れを確認します。 |
| シーケンシャル アイオー オンリー<br>Sequential I/O only | 58 | シーケンシャルファイルに対して、ランダムファイルの扱いをした。 | ファイルの属性を確認して正しく扱うか、属性を変更します。 |
| ストリング フォーミュラ<br>String formula<br>トゥー コンプレックス<br>too complex | 16 | 文字式が長すぎる。または複雑すぎる。 | 短かい式にわけて、文字の演算を行ないます。 |
| ストリング トゥー ロング<br>String too long | 15 | 文字列の長さが255文字を越えようとした。 | 文字列の長さが255文字以内に収まるようにします。 |
| サブスクリプト アウト オブ レンジ<br>Subscript out of range | 9 | 配列の添字が宣言したときの値を越えた。 | 添字の値を検討するか、配列宣言の値を大きくします。 |
| シンタックス エラー<br>Syntax error | 2 | 入力された命令のつづりがちがっていたり、ない命令を実行しようとした。 | つづりの再確認をします。プログラムの入力を再確認します。 |
| タイプ ミスマッチ<br>Type mismatch | 13 | 文字変数に数値を代入したり、変数の型が合っていない。 | 変数の型を確認します。 |
| アンディファインド ライン ナンバー<br>Undefined line number | 8 | GOTO文やGOSUB文で指定した行番号が存在しない。 | 行番号を確認し、正しい行番号を指定します。 |
| アンディファインド ユーザー<br>Undefined user<br>ファンクション<br>function | 18 | DEF FN文で定義していないユーザー関数を使おうとした。 | DEF FNで関数を定義するか、そのような関数を利用しないようにします。 |
| アンプリンタブル エラー<br>Unprintable errors | 23<br>(26~49)<br>60~255 | メッセージが定義されていない。 | ユーザーが自由にエラーコードを使うことができます。ただし26~49は将来BASICを拡張したときに使うのでユーザーは使えません。 |
| ヴェリファイ エラー<br>Verify error | 20 | カセットテープの上のプログラムとメモリー上のプログラムとで内容が異なっている。 | 音量などを変えてもう一度行なうか、新たにセーブしなおします。 |

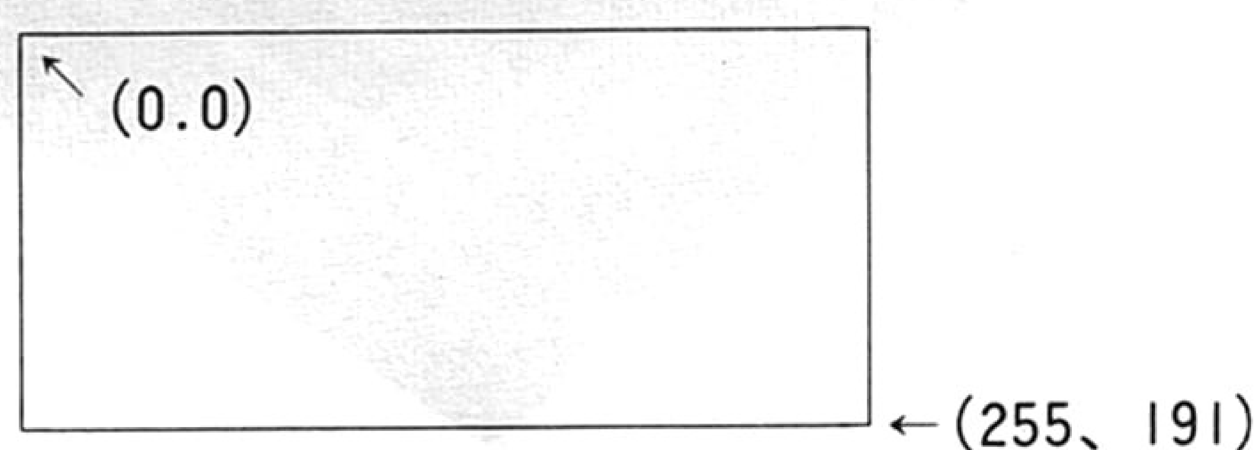
## (8)用語解説

### ●オフセット

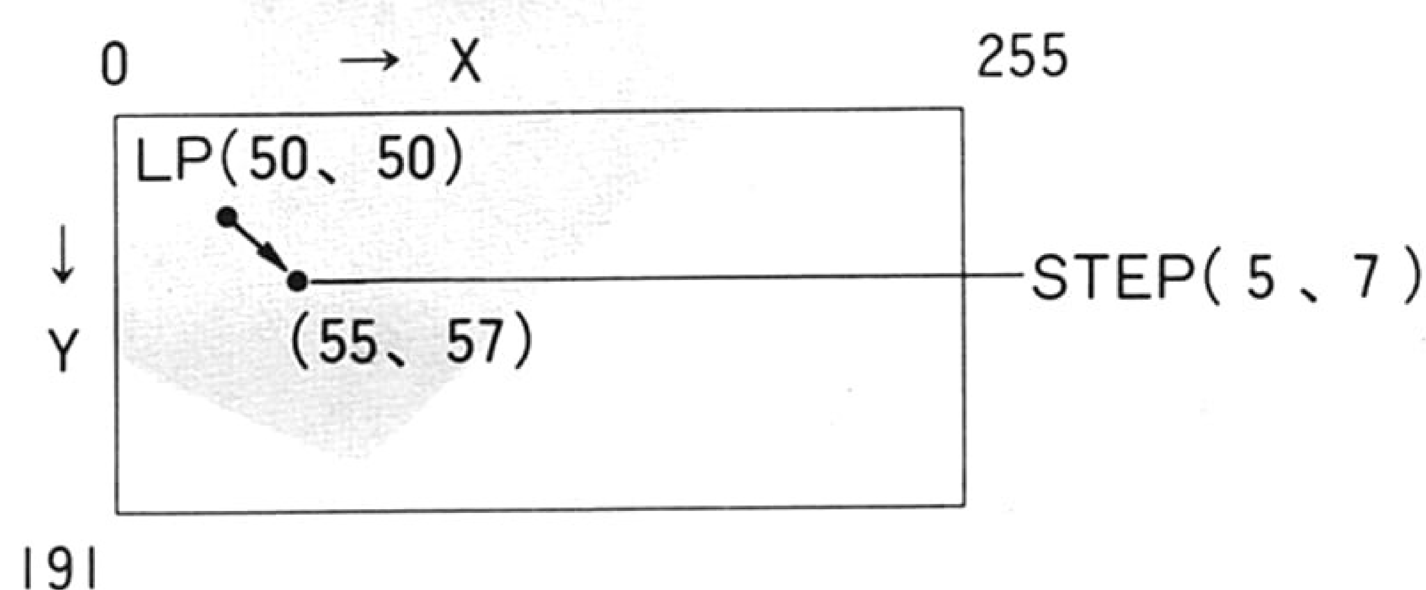
基準となる値からの差のことです。また差の値をオフセット値と呼びます。
BLOADで使うオフセットは、機械語プログラムをセーブしたときは異なる番地にロードしたいときに使います。このときの基準値は、セーブしたときの開始番地で、オフセットは基準値とロードするときの番地との差になります。

### ●絶対座標/相対座標

画面は、横256ドット、縦192ドットから構成されています。この中のひとつの位置を指定する方法が座標です。
横256ドットはX座標で0～255で表わされ、縦192ドットはY座標で0～191で表わされます。




任意の位置は(X座標、Y座標)という形で表わします。このようにX座標とY座標の値により、一意的に決まる座標の指定を〝絶対座標〟と呼びます。
相対座標は、ひとつの座標をもとに、X方向、Y方向にどれだけ移動しているか(相対移動)により決まります。たとえば、基本となる位置を(50、50)としたとき、X方向に(+5)移動し、Y方向に(+7)移動した位置を、相対座標でSTEP(5、7)と表現します。このようにある位置から、どれだけ離れているかで表わす座標の指定を〝相対座標〟と呼びます。




基準となる点は最終参照点(LP)と呼ばれ、グラフィック命令を実行する都度、変更されます。

### ●ポート番号

コンピュータにはプリンタやディスクなどの外部機器がつながっています。これらの外部機器はひとつひとつに番号がわりふられています。これらの番号をポート番号と呼びます。

### ●コマンドレベル

BASICが起動されると〝Ok〟を表示してキー入力待ちとなります。この状態をコマンドレベルと呼びます。BASICでSTOPやENDが実行されたり、エラーが発生して、プログラムが中断されたときにもコマンドレベルになります。

## (9)MX-101のひろがり

MX-101のオプションには、次のものがそろっています。MX-101をシステムアップして使うときの参考にしてください。

**●データレコーダー(KR-10)**

市販のテープソフトを使うときや、自分でつくったプログラムなどを保存するときに使います。KR-10を使うときには、あわせてカセットインターフェイス(FA-32)が必要です。




**●ジョイスティック(TJ-7)**

ゲームを本格的に楽しむときには、このジョイスティックを使いましょう。TJ-7はMSX規格タイプBで、トリガーボタンが2コついています。



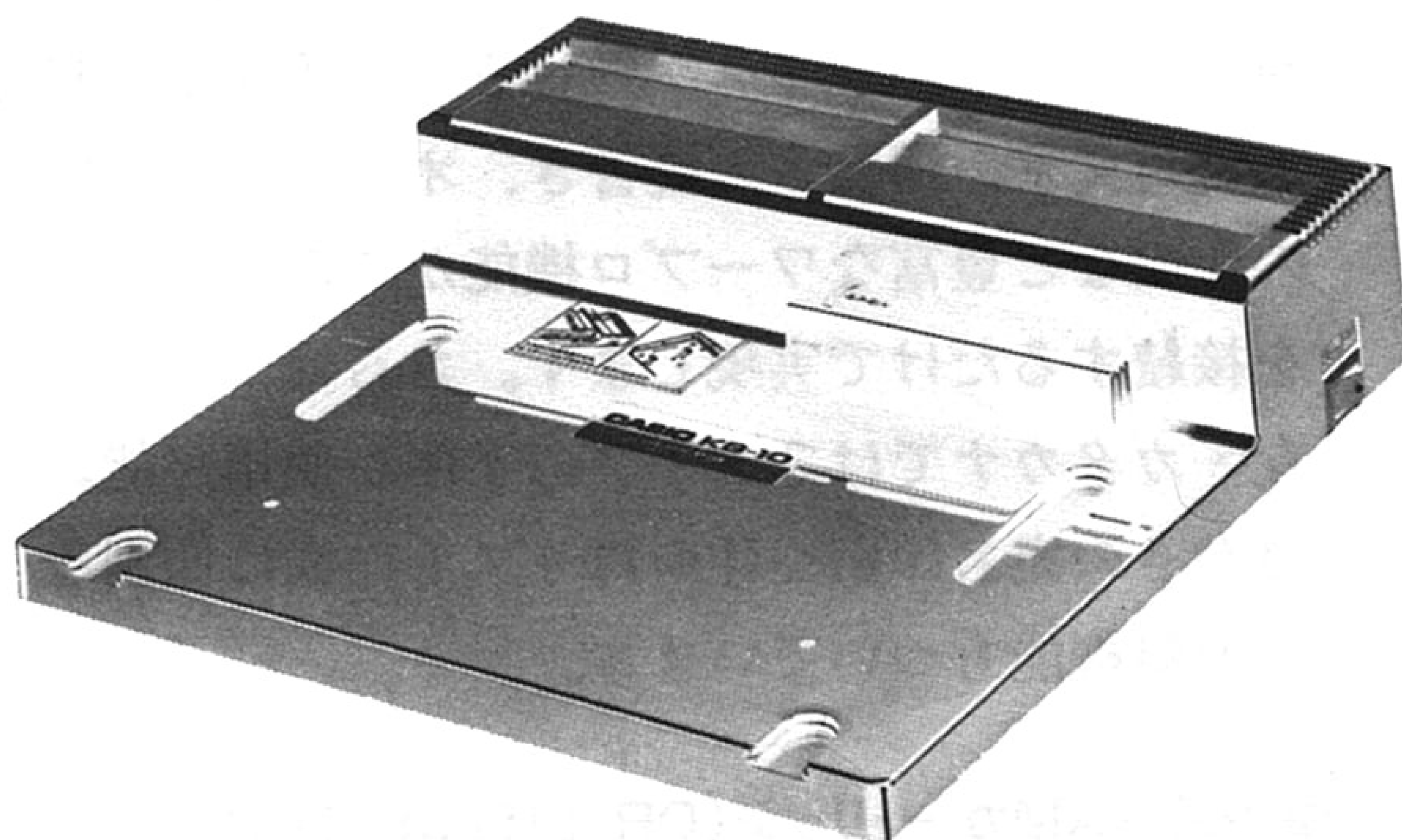

**●タッチパネル(TP-7)**

手描きで文字や絵を入力し、それをテレビ画面に表示させることができるのが、タッチパネルです。カシオのROMカートリッジ「ゲームランドスペシャル」(GPM-501S)や、カシオ「描きくけコン」(GPM-503)でグラフィックに挑戦するときに、とても便利です。



●拡張ボックス(KB-10)

2つのカートリッジスロット、プリンタインターフェイス内蔵の拡張ボックスを使えば、MX-101の使い道がさらに広がります。カラーグラフィックプリンタ（後述）の使用や、RAM容量の増設、また市販ソフトの組み合せなどが可能になります。




●カラーグラフィックプリンタ(CP-7)

16色のカラープロッタプリンタです。自作のプログラムやデータ処理結果の打ち出しができ、また、「描きくけコン」(GPM-503)で描いたグラフィックスもそのままハードコピーがとれます。このCP-7を使うときは、KB-10とプリンタケーブル（PK-7など）があわせて必要です。




●クイックディスク(QD-7)

データ処理や、大容量のプログラムのセーブ／ロードにはクイックディスクが大変便利です。セーブ／ロードの時間は約8秒のスピードです。片面64KB（両面128KB）の記憶容量の2.8インチディスクシート（SD-7）を使います。

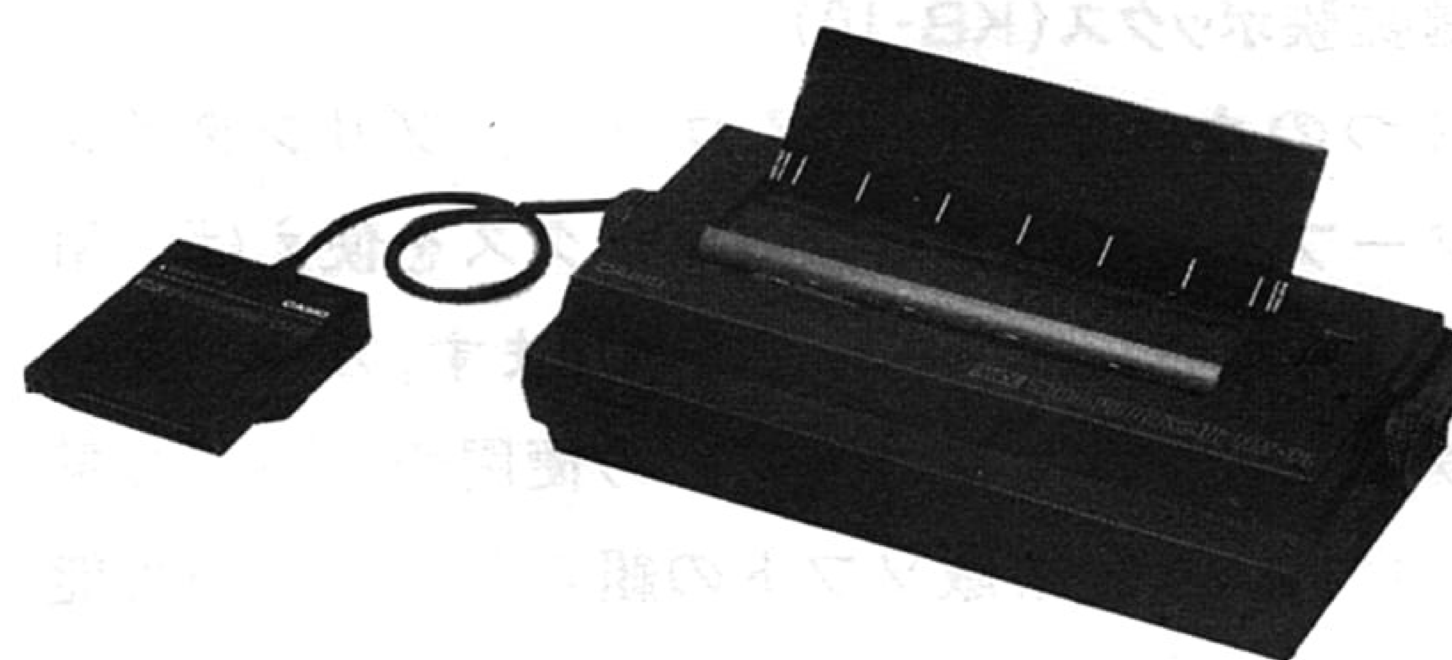



※MX-101本体でも、MX-101とKB-10の接続時も使用できます。

※SD-7は、フォーマット時、片面約58KB(両面約116KB)の記憶容量となります。

**●ワードプロセッサーユニット(MW-24)**

24×24ドットのプリンター付き。半角文字、自由倍角文字、横書き／縦書き、オリジナル文字の作成など豊富なワープロ機能がこのユニットを接続するだけで実現します。さらに、ひらがな・カタカナでは3タイプの字体が使用可能。はがきの宛名印刷や住所録リスト作成に便利な住所録機能もついています。



**●増設RAMカートリッジ(OR-216/OR-264)**

MX-101のRAM容量を増やすときに使用するのがRAMカートリッジです。

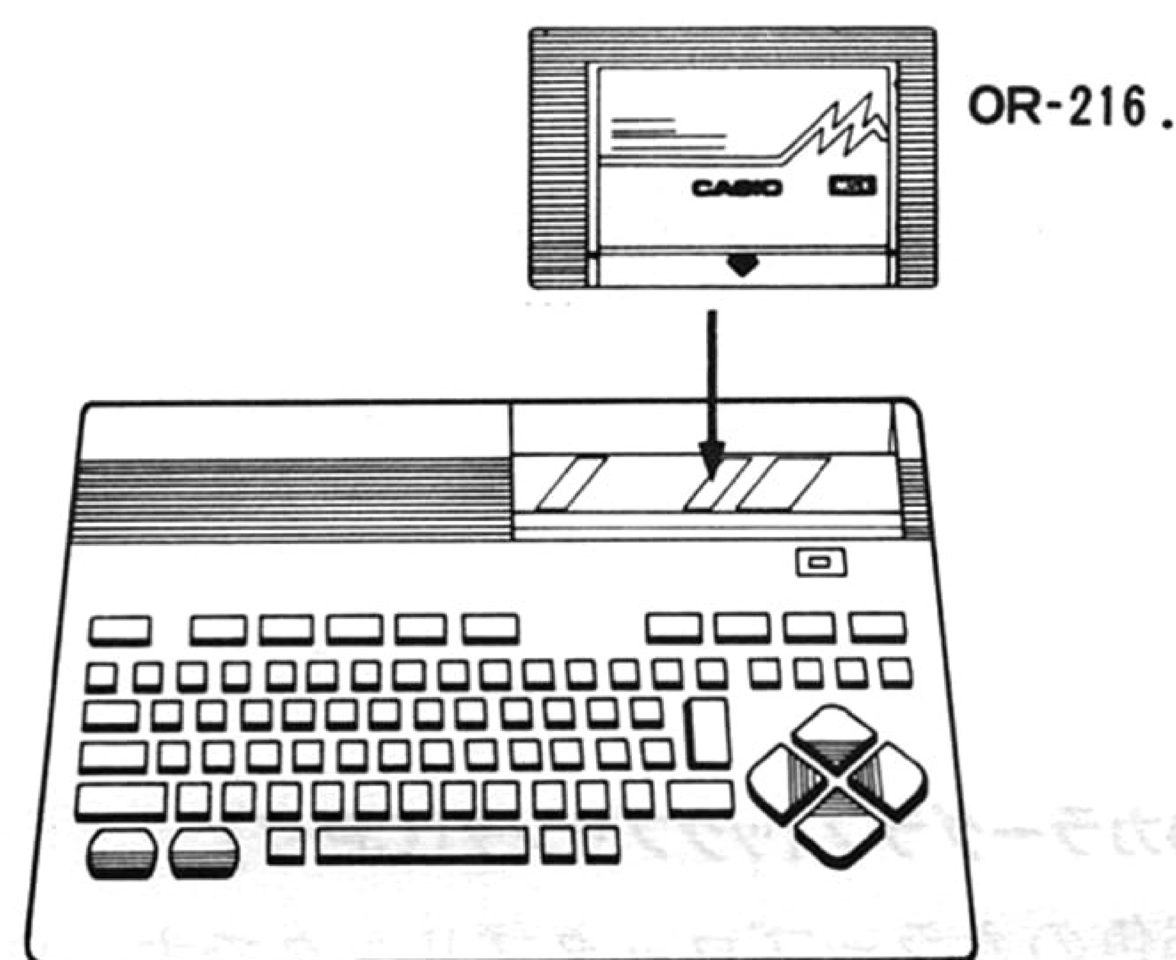
OR-216…MX-101のRAM容量を32KBに増設するときに使います。

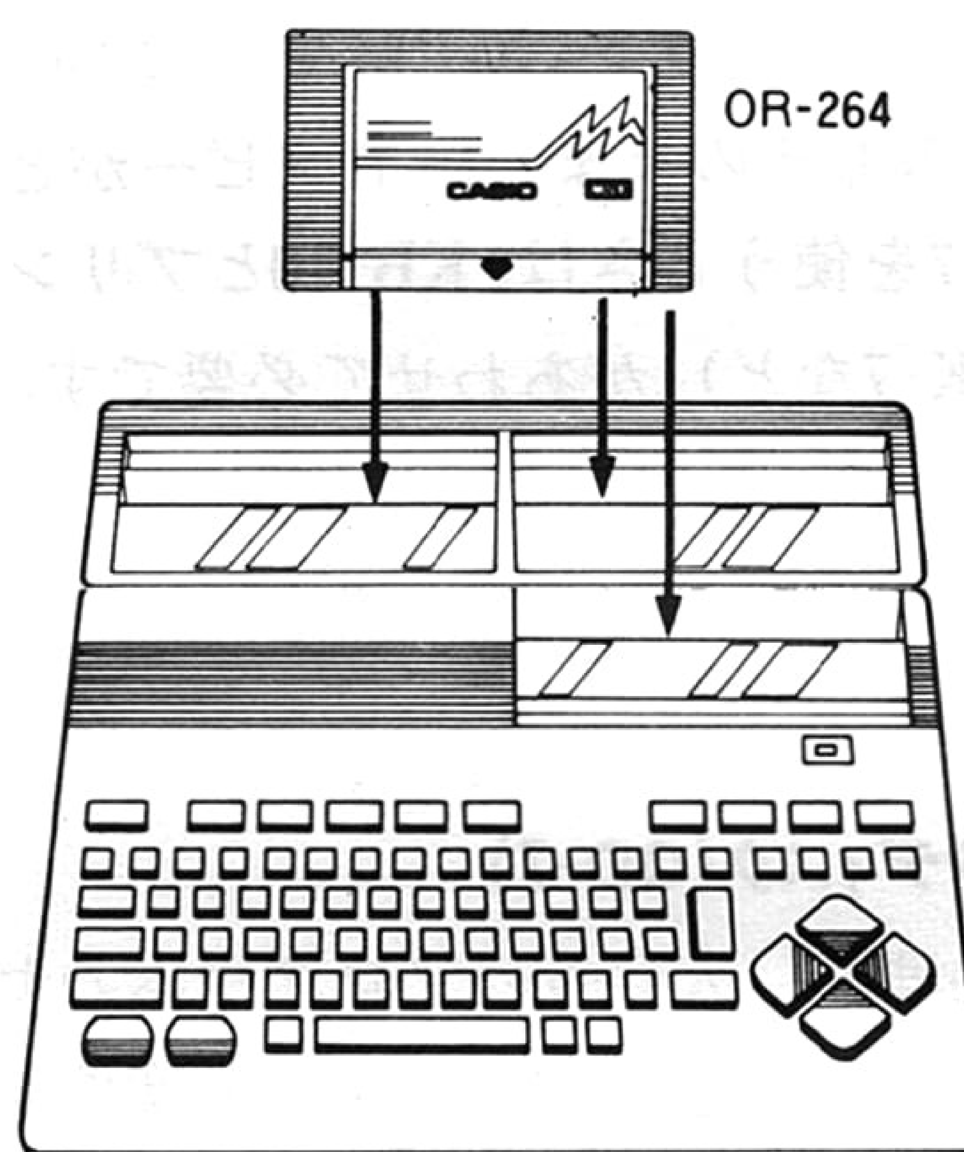
MX-101本体のスロットに装着します。

※MX-101にKB-10を接続した時も使用できます。
ます。

OR-264…MX-101のRAM容量を64KBに増設する時に使います。MX-101にKB-10を接続した状態でスロットに装着して使用します。

※RAM64KBは、市販ソフト使用時に有効で、自分でプログラムをつくる時(MSX-BASIC使用時)は、32KBとなります。




**●カセットインタフェイス(FA-32)**

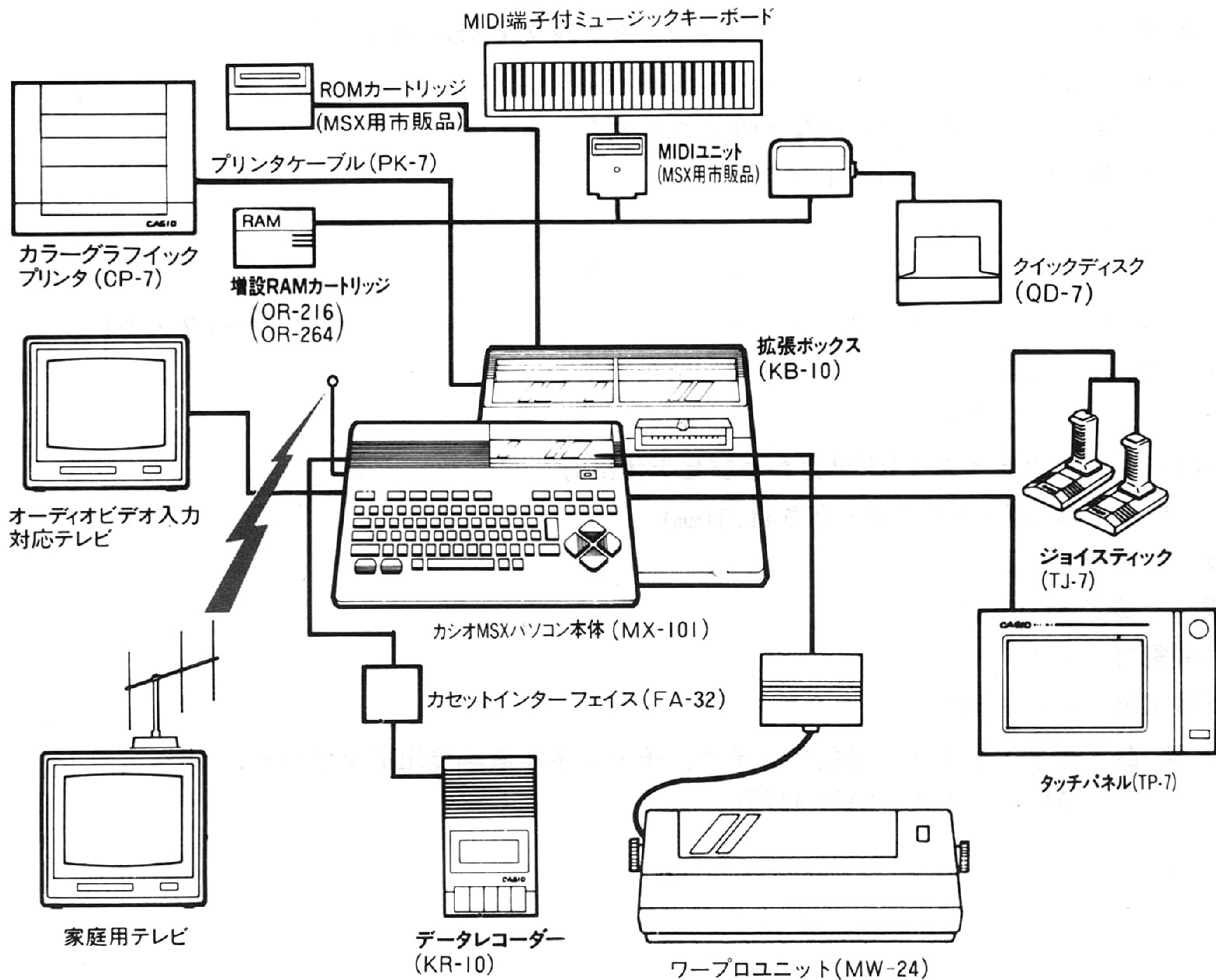
MX-101で作ったプログラムをカセットレコーダーで録音するとき、ゲームテープなどをカセットレコーダーから読み込むときに使います。



それぞれのオプションの詳しい仕様や接続方法は、オプションの取扱説明書をお読みください。

**※掲載の機種の中には品切れになる製品もありますので、販売店でお確かめのうえ、お選びください。**

# システム構成図




※周辺機器との接続については、販売店またはカシオ・サービスセンターにご相談ください。
※仕様および外観、システム構成は、改良のため予告なく変更されることがあります。

## MX-101の仕様

**出力レベル**：90±4dBμV(出力インピーダンス75Ω)
**出力可変範囲**：470.25MHz～478.25MHz(13ch～14ch)
**C P U**：Z80Aコンパチブル（3.58MHz）
**R O M**：32KB（MSX BASIC ROM）
**R A M**：16KB
**V R A M**：16KB
**文字表示**：横32×タテ24文字（最大横40×タテ24文字〈SCREEN (0)〉）
**文字構成**：８×８ドットマトリックス文字
**文　字**：英数文字・ひらがな・カタカナ・グラフィック記号
**ドット数**：横256×タテ192ドット
**色　数**：16色
**サウンド**：８オクターブ、３重和音、ノイズ音
**キー総数**：キー・ファンクションキー・ジョイパッド（８方向)・トリガー(２入力)
**キー配列**：英数字、記号——JIS配列準処
かな文字————50音配列
**CRTインターフェイス**：RF出力およびビデオ出力
**大きさ**：幅257×奥行200×高さ44.3(mm)
**重　さ**：1000g
**電　源**：AC 100V
**消費電力**：８W
**使用温度**：０℃～40℃
**付属品**：送信アンテナ、受信アンテナ、チャンネル微調整用ドライバー、
ACアダプター(AD-4175)

# ▶ 索　引 ◀

# カシオMSXソフト・ライブラリー

## どれから挑戦しようか。よりどりみどりに遊べる。




**大障害競馬**
馬の走りと、レースのかけひきが堪能できる。
GPM-101 ©CASIO ¥4,800




**アイスワールド**
氷の国で、かわいい白クマ"クッキー君"が大活躍。
GPM-112 ©CASIO ¥4,800




**イーグルファイター**
迫力のジェット音と空中戦の興奮がリアルに迫る。
GPM-113 ©CASIO ¥4,800




**カシオワールドオープン**
世界各地の難コースに挑戦しよう。
GPM-114 ©CASIO ¥4,800

★




**伊賀忍法帖**
必殺忍法がいっぱい！オモシロ忍者アクション。
GPM-117 ©CASIO ¥4,800

★




**カーファイター**
めまぐるしいカーチェイスでアメリカ大横断。
GPM-118 ©CASIO ¥4,800




**エグゾイド-Z**
銀河宇宙のエイリアンを倒せ！大興奮の変身ロボアクション。
GPM-120 ©CASIO ¥4,800

★




**イーガー皇帝の逆襲**
熱烈奮闘！痛いほど楽しいイーアールカンフー2。
GPM-121 ©Konami 1985 ¥4,800

★




**魔城伝説**
愛の剣士と地獄の魔王ヒュドノスの棲絶な死闘！
GPM-122 ©Konami 1986 ¥4,800




**妖怪屋敷**
暗い屋敷にお化けがうじゃうじゃ。おっとろしい肝だめし。
GPM-123 ©CASIO ¥4,800




**仔猫の大冒険**
かわいい仔猫チビちゃんが、恋人求めて汗と涙の一匹旅。
GPM-124 ©CASIO ¥4,800




**賢者の石**
異次元空間を抜け出せ！正統派ロールプレイングゲーム。
GPM-125 ©CASIO ¥4,800

★



**スペランカー**

地下に眠る黄金を探せ!スリル満点地底探険。

GPM-126
©1985 1986 IREM
Licensed from Broder bund
¥4,900

★



**ツインビー**

2人プレイで楽しめるミラクルシューティング。

GPM-127
©Konami 1986 ¥4,800




**満月城の戦い**

伊賀丸が、甲賀の本拠地満月城に、挑む忍者アクション。

GPM-128
©CASIO ¥4,800

——エグゾイド-Z——

**●エリア-5** GPM-129 ©CASIO ¥4,800

エイリアンを退治せよ! スピード感満点のスペースシューティング。

**●モアイの秘宝** GPM-130 ©CASIO ¥4,800

モアイ像に隠された宝を探せ! 楽しいパズル&アドベンチャー。

## 創る、解ることから、コンピュータと仲良くなれる。




**ゲームランド**

ごきげんだね。自分だけのゲームが創れる。

GPM-501
©CASIO ¥7,800




**ゲームランド・スペャル**

ゲームランドにカセットテープとテクニック集がついたスペシャル版。

GPM-501S
©CASIO ¥8,900




**描きくけコン(グラフィックソフト)**

ノンプログラムで、思い通りに絵が描ける。

GPM-503
©CASIO ¥4,800

★



**BASIC入門**

画面と対話しながら、BASICが解る。

GPM-502
©CASIO ¥5,800




**BASIC入門II**

(プログラミング編)

楽しみながら、プログラムの組み方が解る。

GPM-505
©CASIO ¥5,800




**コンピュータ入門**

コンピュータの動作原理が画面をとおして理解できる。

GPM-506
©CASIO ¥5,800

※掲載のソフトの中には品切れになる製品もありますので、販売店でお確かめのうえ、お選びください。
※★印の商品は数量に限りがありますので、品切れの際はご容赦ください。

# カシオ計算機株式会社営業本部

東京都新宿区西新宿2－6　新宿住友ビル
(〒163)　☎03-347-4811(代表)

# カシオ計算機サービスセンター

| | | | |
|---|---|---|---|
| 旭川 | 0166-23-8580 | 〒070 | 旭川市七条通り8丁目 |
| 札幌 | 011-231-2343 | 〒060 | 札幌市中央区南一条西12丁目 |
| 釧路 | 0154-24-8575 | 〒085 | 釧路市光陽町6－1 |
| 青森 | 0177-22-7466 | 〒030 | 青森市勝田2－1－12 |
| 秋田 | 0188-33-6211 | 〒010 | 秋田市中通り6－1－55 |
| 盛岡 | 0196-46-3393 | 〒020 | 盛岡市上堂2－3－6 |
| 仙台 | 022-227-1404 | 〒980 | 仙台市国分町2－8－14 |
| 山形 | 0236-42-8018 | 〒990 | 山形市あこや町3－14－39 |
| 郡山 | 0249-33-5172 | 〒963 | 福島県郡山市香久池2－11－6 |
| 宇都宮 | 0286-34-0395 | 〒320 | 宇都宮市西大寛2－1－3 |
| 前橋 | 0272-53-3000 | 〒371 | 前橋市元総社町92－5 |
| 水戸 | 0292-25-6985 | 〒310 | 水戸市城南3－10－17 |
| 埼玉 | 0486-66-8567 | 〒330 | 大宮市大成町4－83 |
| 千葉 | 0472-43-1751 | 〒260 | 千葉市登戸町1－21－15 |
| 東京 | 03－862-4141 | 〒101 | 千代田区神田佐久間町2－23 |
| 城西 | 03－341-3321 | 〒160 | 新宿区新宿2－4－3 |
| 多摩 | 0425-23-3531 | 〒190 | 立川市錦町3－2－25 |
| 横浜 | 045-211-0821 | 〒231 | 横浜市中区弁天通り6－85 |
| 新潟 | 0252-87-1155 | 〒950 | 新潟市弁天橋通り3－9－12 |
| 長野 | 0262-28-9360 | 〒380 | 長野市岡田町30－20 |
| 甲府 | 0552-37-6371 | 〒400 | 甲府市城東2－22－11 |
| 静岡 | 0542-81-8085 | 〒422 | 静岡市西中原1－4－35 |
| 浜松 | 0534-64-1658 | 〒435 | 浜松市西塚町325－6 |
| 名古屋 | 052-263-0454 | 〒460 | 名古屋市中区栄4－6－15 |
| 岐阜 | 0582-62-0145 | 〒500 | 岐阜市鷹見町8 |
| 三重 | 0592-27-5066 | 〒514 | 津市上浜町1－251 |
| 富山 | 0764-22-2251 | 〒930 | 富山市白銀町2－1 |
| 金沢 | 0762-37-8511 | 〒920 | 金沢市諸江町下丁93－1 |
| 京都 | 075-351-1161 | 〒600 | 京都市下京区五条通り堀川東入ル |
| 大阪 | 06－362-8181 | 〒530 | 大阪市北区南森町2－1－20 |
| 神戸 | 078-392-2145 | 〒650 | 神戸市中央区伊藤町119 |
| 姫路 | 0792-34-3502 | 〒672 | 姫路市飾磨区下野田767－1 |
| 岡山 | 0862-41-8471 | 〒700 | 岡山市西古松西町9－1 |
| 広島 | 082-273-7111 | 〒733 | 広島市西区己斐本町2－17－24 |
| 高松 | 0878-62-5240 | 〒760 | 高松市亀岡町9－16 |
| 松山 | 0899-23-2255 | 〒791 | 松山市山越5－10－15 |
| 福岡 | 092-411-2939 | 〒812 | 福岡市博多区東比恵2－16－23 |
| 長崎 | 0958-61-8084 | 〒852 | 長崎市宝栄町2－26 |
| 熊本 | 096-367-0650 | 〒862 | 熊本市健軍4－1－5 |
| 鹿児島 | 0992-56-3575 | 〒890 | 鹿児島市上荒田町30－18 |

MX-101オペレーションマニュアル

制作/編集
株式会社　モダン
株式会社　コーラル

発　行
カシオ計算機株式会社

〒163 東京都新宿区西新宿2-6
新宿住友ビル TEL.(03)347-4811

発行日　昭和61年10月　　MO1051061A

乱丁・落丁本はお取替えいたします。

CASIO

# カシオMX-101保証書

(This Warranty is valid only in Japan.)

この製品は、高度な電子技術と完全な品質管理そして厳重な検査行程を経て造られております。しかし、ご使用中万一障害が発生した場合は、裏面記載の内容により修理および調整をいたします。

★ご販売店様へ
この保証書はお客さまへのアフターサービスの実施と責任を明確にするものです。贈答品、記念品の場合も含めて必ず記入捺印してお客さまへお渡しください。

| | | |
|---|---|---|
| 機種 | | MX-101 |
| 保証期間 | | (お買い上げ日)<br>昭和　年　月　日から本体1ヵ年 |
| お客様名 | | 様 |
| ご住所 | | 〒□□□-□□　TEL. |
| 販売店 | 住所・店名<br>㊞<br>TEL. | |

カシオ計算機株式会社

〒163 東京都新宿区西新宿2-6 新宿住友ビル ☎03-347-4811

## 保証規定

1. 説明書の注意に従った正常な使用状態で故障した場合は、お買い上げ後1年間無料で修理いたします。

2. 修理の必要が生じた場合は、製品に本証を添えて、お買い上げ店または最寄りのカシオ計算機サービスセンターへご持参またはご郵送ください。

3. 修理ご依頼品のご持参およびお持ち帰りの場合の交通費など、またご郵送される場合の郵送料金および諸掛りはお客様のご負担となります。
   ※ご郵送には適切な梱包の上、紛失などを防ぐため簡易書留をご利用ください。

4. 保証期間内でも次の場合は有料修理になります。
   イ. 誤用・乱用および取り扱い不注意による故障
   ロ. 火災・地震・水害および盗難などの災害による故障
   ハ. 不当な修理や改造および異常電圧に起因する故障
   ニ. 使用中に生じたキズなどの外観上の変化
   ホ. 消耗品および付属品の交換
   ヘ. 記憶されているプログラムおよびデータの消失
   ト. 本証の提示がない場合および必要事項(お買い上げ日、販売店名など)の記入がない場合

5. 修理内容などの記録は修理伝票にかえさせていただきます。

6. 本証は日本国内においてのみ有効です。また本証の再発行はいたしませんので大切にご保存ください。

※この保証書によってお客様の法律上の権利を制限するものではありません。

料金受取人払

郵便はがき

163

新宿局承認

3471

差出有効期間
昭和62年10月
15日まで
切手を貼らずに
おだしください。

（受 取 人）
**東京都新宿区 新宿住友ビル内**
**私書箱 25号**

**カシオ計算機株式会社**
**電卓企画部 ご愛用者カード係行**

| | |
|---|---|
| ご住所 | 〒 |
| お名前 または 会社名 | 　 \| 電話 |

**ご購入者**

| 性別 | 男・女 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 年令 | 1 10以下 | 2 11～15 | 3 16～19 | 4 20代 | 5 30代 | 6 40代 | 7 50代 | 生年月日 M・T・S 年 月 日 |

| ご職業 | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 会社員事務職 | 会社員営業職 | 会社員技術職 | 官公庁事務職 | 官公庁技術職 | 商店 | サービス業 | その他自営 | 主婦 | 小学（ ）年 | 中学（ ）年 | 高校・高専生 | 大学・大学院生 | 教師 | その他（ ） |

| | |
|---|---|
| ご使用者 | 1. ご購入者本人 2. ご家族（男・女 才） 3. その他（男・女 才） |
| お買い上げ年月日 | 年 月 日 \| お買い上げ店所在地 |
| お買い上げ店名 | \| 都府道県 市区郡 |
| ご感想欄 | |

**お願い** このカードはご愛用者カードでございます。どうぞ、それぞれの項目にご記入の上ご投函ください。
ご記入いただきましたお答は、企画面に生かし、よりよい商品の開発に役立たせていただきます。

# カシオMX-101

## ❶ 入手方法

1 自分で買った
2 贈り物 どなたから（　　）
3 その他（　　）

## ❷ お買いあげ店は？

1 パソコンショップ
2 玩具店
3 デパート（玩具売場）
4 デパート（文具売場）
5 文具・事務機店
6 電機店
7 カメラ店
8 スーパー（玩具売場）
9 スーパー（文具売場）
10 ディスカウントショップ
11 通信販売
12 その他（　　）

## ❸ お買いあげ価格は

（　　　）円

## ❹ 本機をお知りになったのは？

1 店頭で見て
2 友人に聞いて
3 新聞・雑誌広告で
新聞名・雑誌名（　　・　　）
4 テレビ・ラジオの広告で
5 カタログ・ポスターを見て
6 チラシを見て
7 その他（　　）

## ❺ お買いあげにあたって

●実際に操作しましたか
1 はい
2 いいえ

●他メーカーと比較しましたか
1 はい
・ソニー　・東芝
・松下　・任天堂
・セガ
・その他（　　）
その機種名は（　　）
2 いいえ

●本機を選んだ理由は
1 MSX仕様だから
2 ワイヤレスだから
3 RAM容量16KBだから
4 BASICの勉強のために
5 ゲームで遊べるから
6 拡張性にすぐれているから
7 ゲームが創れるから
8 グラフィックスが出来るから
9 小型・軽量だから
10 デザイン・色がよいから
11 価格が手頃だから
12 カシオの製品だから
13 友人・知人のすすめで
14 その他（　　）

●本機の主な用途は何ですか
1 ゲームで遊ぶ
2 ゲームを創る
3 パソコンの勉強をする
4 コンピュータグラフィックスの作成
5 コンピュータミュージックの演奏
6 ワープロとして使う
7 実務で使用する
8 教育に使用する
9 趣味に利用する
10 その他（　　）

## ❻ カシオのオプションをお持ちですか

1 持っている
(A)ジョイスティック（TJ-7）
(B)拡張ボックス（KB-10）
(C)増設16KB RAMカートリッジ（OR-216）
(D)増設64KB RAMカートリッジ（OR-264）
(E)タッチパネル（TP-7）
(F)データレコーダ（KR-10）
(G)カラーグラフィックプリンタ（CP-7）
(H)クイックディスク（QD-7）
(I)カセットインターフェイス（FA-32）
(J)ワープロユニット（MW-24）
2 持っていない

●今後、購入予定のものは
(A) (B) (C) (D) (E) (F) (G) (H) (I) (J)

## ❼ お使いになってみての感想は

●性能は
・満足している
・不満である
・どちらとも言えない

●本体の容量は
・充分である
・もの足りない
・どちらとも言えない

●操作性は
・使いやすい
・使いにくい
・どちらとも言えない

●形状は
・ちょうどよい
・もっと小型がよい
・もっと大型がよい

●マニュアルは
・わかりやすい
・わかりにくい
・どちらとも言えない

MO1051061A　　ご協力ありがとうございました。

CASIO®